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

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

В.о.завідувача кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

на тему: «Виявлення аварійних ситуацій на підприємстві шляхом впровадження технологій інтернету речей»

Виконала:
студентка IV курсу, групи КА-64
Хархута Марія Сергіївна _____

Керівник:
Асистент Кухарєв Сергій Олександрович _____

Консультанти:
Оргінізаційно-економічна частина доц. Шевчук О.А. _____
Нормоконтроль к.т.н., доц. Коваленко А.Є. _____

Рецензент:
Асистент Безносик О.Ю. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління

ЗАТВЕРДЖУЮ

В.о.завідувача кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Хархута Марія Сергіївна

1. Тема роботи «Виявлення аварійних ситуацій на підприємстві шляхом впровадження технологій інтернету речей», керівник роботи Кухарєв Сергій Олександрович асистент, затверджені наказом по університету від «_25_»_травня_ 20_20_ р. №_1143-с__

2. Термін подання студентом роботи 08 травня 2020 року

3. Вихідні дані до роботи

Платформа інтернету речей, яка:

- Дозволяє ідентифікувати аварійну ситуацію на підприємстві
- Вирішує основні задачі, притаманні подібним платформам
- Має гнучку архітектуру, яку можна адаптувати під аналітику даних

4. Зміст роботи

1. Розглянути предметну область та існуючі рішення, виділити шляхи їх покращення.

2. Сформулювати функціональні та нефункціональні вимоги до системи.

3. Розробити архітектуру системи; описати протоколи та формати обміну даними між компонентами.

4. Виділити вимоги до складових частин системи; обґрунтувати вибір сторонніх рішень.

5. Виконати програмну реалізацію платформи. Описати важливі аспекти її реалізації.

7. Навести приклад використання системи.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)
14 рис + презентація

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Оргінізаційно-економічний	доц. Шевчук О.А.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	18.04.2020	виконано
2	Збір інформації	19.04.2020	виконано
3	Дослідження продметної області, формулювання вимог до системи	22.04.2020	виконано
4	Розробка архітектури системи	26.04.2020	виконано
5	Виконання програмної реалізації платформи	28.04.2020	виконано
6	Вибір сторонніх рішень	11.05.2020	виконано

7	Розробка прикладу використання платформи	01.06.2020	виконано
8	Оформлення дипломної роботи	02.06.2020	виконано

Студент Хархута М.С.

Керівник Кухарєв С.О.

РЕФЕРАТ

Дипломна робота: 82 ст., 18 рис., 9 таб., 3 дод., 41 джерел.

ІНТЕРНЕТ РЕЧЕЙ, РОЗПОДІЛЕНА СИСТЕМА, ОБРОБКА ВЕЛИКИХ ОБСЯГІВ ДАНИХ, МОНІТОРИНГ МЕРЕЖІ ПРИСТРОЇВ, APACHE SPARK, APACHE KAFKA, APACHE CASSANDRA, MONGODB.

Об’єкт дослідження – дані, які генерує розумний датчик.

Мета роботи – дослідження мікросервісних систем, що забезпечують роботу Інтернету речей та розробка платформи, яка дозволяє виявити аварійну ситуацію на підприємстві.

Методи дослідження – проаналізовано протоколи та методи, що використовуються для передавання інформації в мережі інтернету речей, що дало змогу виявити позитивні та негативні якості кожного з протоколів, та обрати оптимальні для створення мережі. Розглянуто побудову платформи за допомогою новітніх підходів до розробки розподілених систем прийому, зберігання та обробки великих обсягів даних. Для побудови платформи використано сучасні бібліотеки та бази даних із відкритим вихідним кодом.

Запропоновано архітектуру платформи, яка забезпечує високу масштабованість і ефективність, а також дозволяє реалізувати рішення для моніторингу мережі пристроїв і роботи з адміністративними даними, які є більш функціональними за існуючі.

Практичне завдання: використання платформи продемонстровано на прикладі.

ABSTRACT

Thesis: 82 pp., 18 pic., 9 tab., 3 add., 41 sources.

INTERNET OF THINGS, DISTRIBUTED SYSTEM, BIG DATA PROCESSING, DEVICE NETWORK MONITORING, APACHE SPARK, APACHE KAFKA, APACHE CASSANDRA, MONGODB.

The object of research is the data generated by a smart sensor.

The purpose of the work - the study of microservice systems that provide the Internet of Things and the development of a platform that allows you to detect an emergency situation at the enterprise.

Research methods - analyzed the protocols and methods used to transmit information on the Internet of Things, which allowed us to identify the positive and negative qualities of each of the protocols, and choose the best to create a network. The construction of the platform using the latest approaches to the development of distributed systems for receiving, storing, and processing large amounts of data is considered. Modern open-source libraries and databases were used to build the platform.

The platform architecture is proposed that provides high scalability and efficiency, as well as allows the implementation of solutions for monitoring the network of devices and work with administrative data that are more functional than existing ones.

Practical task: the use of the platform is demonstrated by example.

ЗМІСТ

ВСТУП

Інтернет речей (IoT) відноситься до мережі розумних пристроїв, які можуть постійно взаємодіяти зі своїм оточенням. Ці пристрої здатні спілкуватися та реагувати на інформацію, яку вони збирають, дозволяючи системі полегшувати діяльність, упорядкувати процеси та інформувати прийняття рішень. Продукти IoT стали більш популярними і помітними в сучасному суспільстві у вигляді додатків, таких як: побутова інтелектуальна техніка, транспортні системи або автоматизація складів. Платформа Інтернету речей має забезпечувати можливість аналізувати різні аспекти даних, що потрібно для оптимізації різноманітних виробничих та інших процесів.

Основна особливість Інтернету речей - величезна кількість взаємозв'язків між різними пристроями та доступ до глобальної мережі. З цієї причини пристрої IoT додатково потребують інфраструктури, яка забезпечує безперебійний дротовий та бездротовий зв’язок з високою пропускною здатністю, низькою затримкою та постійним доступом до Інтернету.

У цій роботі буде розглянуто використання хмарної платформи інтернету речей для виявлення аварійних ситуацій на підприємстві.

Також проводиться аналіз технологічних та архітектурних рішень в секторі інтернету речей, вивчаються підходи програмного забезпечення. Така послідовність дає змогу задовільнити необхідні вимоги до системи, розробити сучасну та прогресивну платформу.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ. МЕТА РОБОТИ

1.1 Предметна область і актуальність роботи

Інтернет речей (Internet of Things – IoT) рішення які допомагають доставляти та керувати різними частинами, такими як: пристрої, датчики, безпека, мережі та платформи [1].

IoT позначає постійно зростаючу зв’язок повсякденних предметів, які можуть відчувати та взаємодіяти один з одним та з оточенням. Обстеживши різні визначення, використовувані галузями та групами інтересів у 2015 році, Інститут інженерів електротехніки та електроніки (IEEE) запропонував просте визначення поняття "мережа, яка з'єднує однозначно ідентифіковані" речі "з Інтернетом", де "речі" - це пристрої, які можуть відчувати та взаємодіяти відповідно до своїх апаратних та програмних можливостей[2].

Датчики надають щоденним пристроям та об’єктам розумну, підключену функціональність, і багато пристроїв Інтернет речей тепер розроблені за допомогою декількох датчиків для збору різноманітних даних, за якими можна збирати, аналізувати та діяти. Заходи безпеки, що захищають пристрої та мережі IoT від атак, частіше вбудовуються в основні операції пристроїв через архітектуру безпеки. Платформа Інтернет речей - це гнучка, безпечна та ефективна основа, яка охоплює зв’язок , пристрій та управління даними[1].

Щоб йти в ногу зі швидким розвитком нових пристроїв IoT, компанії використовують переваги IoT-рішень, які пропонують в режимі реального часу багаті операційні системи, пристрої малої потужності та вбудовані функції безпеки. Рішення IoT можуть включати, більш конкретно, рішення технологій IoT, рішення керування пристроями та рішення безпеки. Ця платформа є основним рушієм цифрової трансформації, що дозволяє підприємствам винаходити продукти, послуги, внутрішні операції та бізнес-моделі. Рішення IoT дозволяють компаніям легко використовувати переваги як надійну, безпечну та потужну сітку підключених пристроїв, будівель та інфраструктури[3].

Системи IoT торкаються майже всіх аспектів життя. Тому корисно класифікувати ці системи за їх застосуваннями. Один простий і широкий спосіб зробити це сортування залежно від того, чи пристрій чи система IoT призначені для споживчого або промислового використання. У наведеному нижче списку представлений корисний, але не вичерпний опис категорій пристроїв IoT. Наприклад, ці категорії не включають автомобілі, що керують автотранспортом, вбудовані медичні пристрої чи інші потенційні системи, які були б класифіковані в межах IoT. Розумні носії, такі як трекери, які відстежують показники здоров'я, схеми сну або спортивної діяльності. Також цифрові пристрої для спілкування, навігації чи розваг.Пристрої Smart Home для освітлення, опалення та кондиціонування, безпеки або санітарії[3].

Серія фактів із технології Інтернет речей[3]:

- проекти Smart City, такі як ініціативи щодо розумної інфраструктури (наприклад, контроль руху, розумна парковка, поводження з відходами).
- Розумні системи навколишнього середовища, які вимірюють якість повітря, води, контролюють клімат, полегшують управління сільського господарством або підтримують запобігання катастрофам.
- Smart Enterprise включає загальні функції для промисловості, такі як логістика та транспорт, автоматизація товарних ліній або управління запасами роздрібної торгівлі [4].

Рішення IoT також може створювати безперебійну комунікацію між пристроями та працівниками. Їх використання буде лише продовжувати зростати; Очікується, що до 2030 року до Інтернету буде підключено 500 мільярдів пристроїв, кожен з яких збиратиме дані, взаємодіятиме з навколишнім середовищем та спілкуватиметься через мережу. Цей передбачуваний результат повинен заохочувати розробників та користувачів якнайкраще усвідомлювати та підготовитись до потенційних шляхів, яким можуть загрожувати суспільні інтереси. Важливі питання щодо підзвітності та прозорості, що стосуються питань безпеки та конфіденційності, повинні бути включені до розмов про політику[4].

Існує ряд випадків використання, коли пристрої IoT можуть грати роль в аварійних ситуаціях[4]:

1.Моніторинг напруги мережі;

2.Справність роботи механічних пристроїв;

3.Контроль показників стану середовища на підприємстві;

Особа, яка має доступ до інформації, зокрема, отримана з IoT платформи, може надати необхідні дані, щоб нейтралізувати аварійну ситуацію, напривити спеціалістів на місце[4].

1.2 Мета роботи. Функціональні та нефункціональні вимоги до системи

Користуючись перевагами IoT, підприємства стикаються з багатьма проблемами, включаючи інтеграцію IoT-інфраструктури в існуючі системи, розуміння незнайомих форматів даних та протоколів зв'язку, а також впровадження нових технологій у континуумі IoT. Навігація серед цих проблем вимагає ретельного планування, знань про домен та суворої реалізації. Для того, щоб ініціативи IoT були успішними, існує п'ять основних вимог до процесів та практик, які повинні враховувати організації:

- Крайові обчислення - технологія, яка, як очікується, зросте з високою 40% в Азіатсько-Тихоокеанському регіоні до 2023 року, фіксує та аналізує дані про розподілені пристрої, розташовані на межі мережі. Він включає як локальні датчики, які збирають дані, так і крайові шлюзи, які їх обробляють. Крайові обчислення дозволяють проводити аналіз даних поблизу місця, де вони захоплені, що призводить до швидшого реагування на зміни умов. Насправді система обробки краю може реагувати за кілька мілісекунд, порівняно з хмарною системою, яка може зайняти більше 100 мілісекунд [5].
- Прийом даних та обробка потоків - шість із 10 керівників інформаційних технологій кажуть, що збирання, зберігання, інтеграція та аналіз даних у реальному часі з пристроїв кінцевих точок є ключовим бар'єром для успішної реалізації IoT. Організації повинні встановити процеси для збору даних з декількох пристроїв та

датчиків та перетворення їх для використання на хмарних аналітичних платформах. Введення даних стосується імпорту та перетворення даних телеметрії пристрою у формат, придатний службам IoT. Це допомагає нормалізувати дані у загальній моделі даних, яку простіше аналізувати бізнес-додатки та користувачі. Прийом даних також стає корисним, коли організації доводиться забезпечувати збереження поглинутих даних відповідно до урядових або галузевих норм, таких як Загальний регламент Європейського Союзу про захист даних або Закон про захист персональних даних у Сінгапурі [6].

- Безпека та управління пристроями - при швидкому розповсюдженні датчиків IoT та зростаючій складності та обсязі обміну даними організаціям вкрай важливо посилити їх прийняття та впровадження високорозвинених практик та процедур безпеки. Масштаби інвестицій, талант, а також думка про лідерство навколо безпеки потребують різкого зростання, оскільки впровадження IoT зростає та починає ставати основою щоденних операцій в організаціях. Підприємствам потрібно забезпечити безпечне налаштування їхніх пристроїв IoT, ефективну комунікацію та оновлення прискореного та спритного підходу. Управління пристроями охоплює обладнання, програмне забезпечення та процеси, що забезпечують належну реєстрацію пристроїв, управління, захист та оновлення пристроїв [5].
- Розширена аналітика – в даний час широкомасштабна обробка може включати завантаження, що перевищують 100 000 подій в секунду. З прийняттям холодної обробки даних великий обсяг даних аналізується за допомогою вдосконалених алгоритмів після зберігання даних на хмарній платформі. Такий аналіз може виявити тенденції чи коригувальні дії, необхідні для покращення бізнесу чи досвіду клієнтів. На відміну від потокової аналітики ("гарячий шлях"), яка застосовує відносно прості правила до даних в режимі реального часу для короткочасних дій (виявлення шахрайства, порушення безпеки або критичні збої компонентів), холодна обробка шляху включає більш складну аналітику великих даних, наприклад машинне навчання та ШІ, застосовуючись для більш глибокого розуміння[5].

Метою роботи є отримання сигналу про аварійну ситуацію.

Проект вважається виконаним, якщо платформа задовольняє такі умови:

- Ефективно зберігає та оброблює велику кількість даних із різних джерел та відповідає загальним вимогам до системи [7].
- Розробка була виконана з використанням новітніх технологій, вибір яких обґрунтовується [7].
- Платформа своєчасно ідентифікує відхилення у роботі.

На рисунку 1.1 зображена типова архітектура взаємодія IOT платформи та дивайсів.

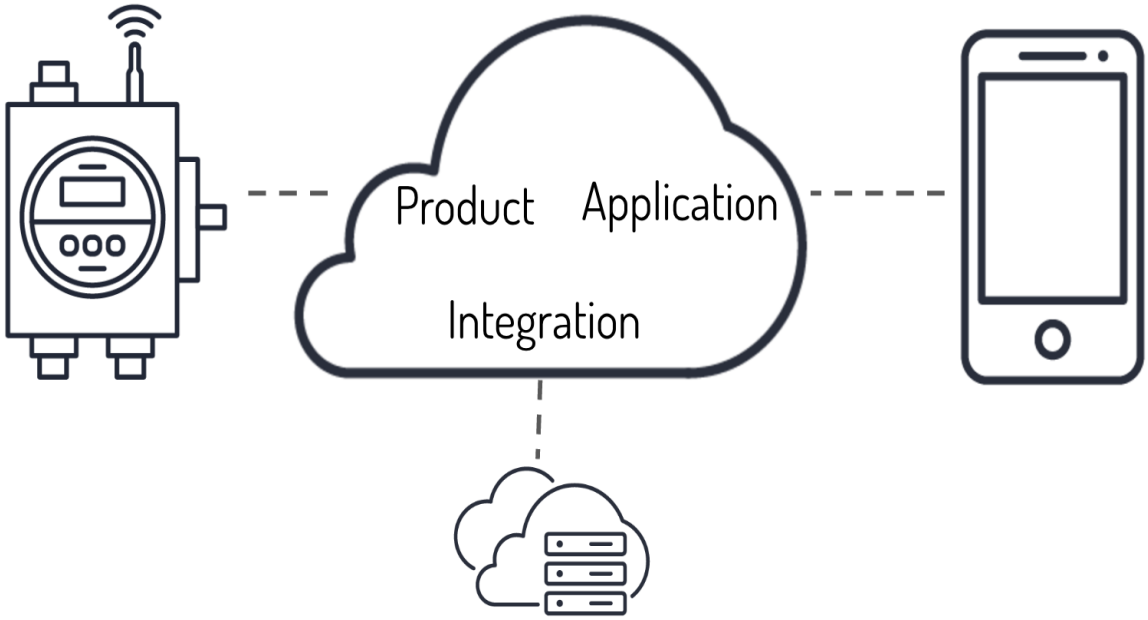


Рисунок 1.1 – Типова архітектура систем інтернету речей

Як правило, в промисловому IoT немає прямого доступу до кінцевих пристроїв, тому шлюзи використовуються для підключення рівнів технологічного обладнання та інтелектуальних систем обробки та зберігання інформації [8].

Кінцеві пристрої - це джерела даних з низькою потужністю обробки, які постійно передають на шлюз багато інформації різних форматів. Датчик кінцевого пристрою генерує аналоговий сигнал, який перетворюється в цифрове (дискретне)

значення за допомогою АЦП - аналого-цифрового перетворювача. Це значення маркується часом та класифікується (позначено) місцевим процесором кінцевого пристрою. Теги можуть бути простими, наприклад, виявленим рухом або складним кількома параметрами (рух + швидкість, рух + швидкість + автомобіль тощо). Чим складніша мітка, тим потужнішим повинен бути периферійний процесор і енергоспоживання кінцевого пристрою. Однак більш інформативні теги дозволяють зменшити кількість переданих у хмару даних та пропускну здатність інформації, а це, у свою чергу, збільшує швидкість реакції на подію [8].

Шлюз, в свою чергу, передає дані в хмарний кластер, де розгорнута програмна платформа IoT, заснована на інструментах Big Data для обробки та видобутку інформації [9].

На хмарному сервері дані з різних периферійних пристроїв інтегруються (узагальнюються за тегами), систематизуються та аналізуються за допомогою машинного навчання та інших методів штучного інтелекту. Результати обміну даними візуалізуються у вигляді графіків, діаграм тощо, що відображаються на інформаційних панелях користувальницького інтерфейсу платформи IoT [10].

Однак Інтернет речей передбачає не тільки передачу інформації від технологічних об'єктів, але і віддалене управління. Тому реалізується зворотний зв'язок від хмарної платформи IoT до пристрою периферійного управління необхідного об'єкта, наприклад, клапана на трубі тощо. Для цього в хмарі реалізується віртуальне подання периферійного пристрою, де записується необхідна інформація про зміну його стану, а потім передається виконавчому пристрою кінцевого обладнання. Будучи периферійним, ЦП виконує розпізнавання тегів і ЦАП, тобто зворотне цифро-аналогове перетворення - від дискретного значення до аналогової форми [11].

Вся система IoT розподілена і масштабована, але вона не пов'язана надійними каналами передачі даних. Тому застосовуються механізми гарантованої доставки інформації. Зокрема, якщо дані не можуть бути передані з кінцевого пристрою в хмару або навпаки, проводяться повторні спроби. Для обміну сигналами між компонентами розподіленої системи використовуються спеціальні рішення -

брокери повідомлень, які гарантують доставку необхідних даних одному або більше одержувачам через керовану чергу [10].

Найпопулярніші брокери повідомлень - RabbitMQ, Apache Qpid, Apache ActiveMQ. Також для цієї мети використовується розподілений реплікуваний журнал змін Apache Kafka, який ідеально масштабується, забезпечуючи збільшення потужностей зі збільшенням кількості та завантаження з джерел даних, а також кількості додатків на їх обробку (абонентів) . Платформа обробки Apache NiFi події (повідомлення) або її спрощена модифікація Apache MiNiFi часто використовується для швидкого завантаження даних з кінцевих пристроїв [11].

Важливим також є сформулювати нефункціональні вимоги до платформи. Для цього скористаємось CAP теоремою.

Проектування розподіленої системи – це завдання, яке вимагає ретельної міркування та обґрунтування. Він вимагає передбачення, накопиченого за роки досвіду та знань про сучасні технології, щоб витрати та зусилля доповнювали кінцевий результат. Особливо слід бути обережними, особливо якщо мова йде про розширення масштабу системи (як вертикально, так і горизонтально) та як система стикається з навантаженням щодо масштабу. Сучасні широкомасштабні системи використовують власні структури даних для досягнення своїх цільових цілей дизайну. Розробка власних протоколів, невідповідність регламенту має важливе значення, оскільки їх інфраструктура не залежить від решти світу [12].

Теорема CAP, яку вперше розробив Ерік Брюер, є одним із способів розрізнити дизайн розподіленої системи: але це не єдиний спосіб. CAP розшифровується як евристика, яку теорема вважає важливою, коли справа стосується проектування а розподіленою системою. Теорема CAP говорить нам про те, що система має володіти наступними якостями[12]:

- Послідовність (Consistency) – розподілені системи відомі для реплікації, тобто вони працюють в кластерах, співпрацюючи один з одним. Не існує поняття глобального сховища пам’яті, і, таким чином, кожен вузол повинен зберігати свою власну копію фрагментів пам’яті, які кластер має намір поділити між собою. Коли в вузлі відбувається оновлення (запис, яке зазвичай походить від клієнта), воно

повинно відображатися на всіх інших вузлах, щоб решта клієнтів, які спостерігають за вузлами (клієнти, пов'язані з іншими вузлами), бачили нові оновлені значення. Система, яка не відповідає цій консистенції, називається непослідовною системою. Однак теорема CAP констатує та дотримується одного конкретного рівня узгодженості, який називається лінійною послідовністю, що вимагає дотримання[13].

Послідовна послідовність системи оновлює репліки, як тільки відбувається оновлення. Це не витрачає часу на оновлення інших вузлів про запис, і, таким чином, можна підтримувати узгодженість. Однак слід зазначити, що така синхронна поведінка є витратою або накладними витратами з точки зору продуктивності, оскільки операції оновлення не повертаються, а наступні операції зчитування блокуються, поки не буде виконана реплікація [14].

- Доступність (Availability) – Як правило, доступність вузла вимірюється його часом роботи. Вузол, що має 90% часу роботи, доступніший, ніж вузол, що має 80%. Однак існує різниця між доступністю та наявністю CAP. Доступність CAP підказує про невдалі вузли, де, якщо вузол не вийшов з ладу, він повинен відповідати на вхідні повідомлення (не має значення, чи є відповіді помилками). Наявність CAP також передбачає необмежений час відповіді. Останнє припущення є менш практичним явищем, оскільки необмежений час відповіді може призвести до втрати груп користувачів. Крім того, практичність диктує, що загалом доступність обговорюється для всієї системи - не лише для кожного невдалого вузла. Високодоступна система має важливе значення, оскільки вона визначає відкриття послуг для користувачів. Наприклад, у високодоступній системі система не надає користувачеві необхідну послугу, хоча деякі вузли вийшли з ладу. Доступність легко досягається тиражуванням вузлів [15].
- Толерантність розділів (Partition Tolerance) – це неминучість того, що мережа іноді виходить з ладу через складності як фізичні, так і логічні. Ми звикли до відмови мереж через фізичні проблеми, такі як погана проводка, але є й інші логічні проблеми, такі як збирання сміття, які можуть спричинити проблеми для мереж. Таким чином, мережі неминуче розподіляються на кілька груп через збої в мережі.

У теоремі CAP толерантність до розділів визначається як здатність враховувати будь-яку втрату повідомлення між розділами. Іншими словами, якщо є два кластери G1 і G2 (розділи), ми можемо припустити, в гіршому випадку, що зв’язок зв'язку між G1 і G2 не вдався. Система повинна мати змогу це враховувати [15].

1.3 Висновки до розділу 1

У розділі розглянуто та проаналізовано архітектуру мережі Інтернету речей. Наведено основні вимоги, котрі застосовують для побудови таких мереж. Платформа повинна бути масштабованою, відмовостійкою та продуктивною

Також розглядаються можливості технологій, які можна використати для побудови платформи.

2 ТЕОРЕТЧНИЙ ОПИС АРХІТЕКТУРИ МЕРЕЖІ ІНТЕРНЕТУ РЕЧЕЙ

Продукти або послуги IoT були впроваджені майже в кожній галузі суспільства. В останні роки все більше компаній виходять на ринок IoT, пропонуючи нові продукти та послуги, які або призначені для підтримки системи IoT, або інтегруються в одну. Мережеві компанії, такі як AT&T, Cisco та Huawei, а також постачальники хмарних платформ, такі як Amazon, Google та Microsoft, вже мають продукти для бізнес-рішень IoT. Виробники компонентів електроніки, такі як Intel та Qualcomm, також пропонують інтегровані платформи IoT як послуги B2B (бізнес для бізнесу). У споживчому просторі виробники пристроїв, такі як Bosch, Google, Amazon, Apple та Samsung, конкурують за чільну частку ринку розумного дому[8].

Проект звіту Національного інституту стандартів і технологій (NIST) описав IoT як два основоположних поняття, які можна об'єднати для отримання визначення[8]:

"Компоненти (деякі з яких можуть мати датчики та пускачі, що дозволяють їм взаємодіяти з фізичним світом), які з'єднані мережею, що забезпечує потенціал для взаємозв'язку між багатьма компонентами"[8].

До важливих особливостей системи IoT належать[9]:

- Вбудовані комп'ютери: пристрої запрограмовані та керовані операційною системою з певною функцією в межах більшої механічної або електричної системи.
- Підключення: пристрої підключені до Інтернету, як правило, через Wi-Fi, Bluetooth, стільникові мережі або інші форми підключення.
- Програмованість: пристрої можуть отримувати оновлення програм або систем, які можуть змінювати, покращувати або змінювати їх функціональність.
- Унікальний ідентифікатор: пристрої повинні бути однозначно ідентифікованими, це означає, що вони можуть відрізнятись та дозволено спілкуватися з людиною чи іншим пристроєм[16].
- Автономія: пристрої можуть вимагати певної форми можливостей прийняття рішень, навіть коли вони відключені від Інтернету чи інших пристроїв.

- Всюдисутність: пристрої, як правило, завжди працюють та доступні.

2.1 Архітектура системи

Інтернет речей (IoT) включає цілу екосистему інструментів та послуг, які повинні об'єднатися, щоб забезпечити повне рішення.

Незалежно від випадку використання, майже кожне рішення IoT включає ті самі чотири компоненти: пристрої, підключення, платформу та додаток. Деякі випадки використання можуть включати додаткові шари, але ці чотири компоненти є основою кожного рішення IoT [17].

Пристрої IoT складають фізичну апаратну складову рішення. У випадку використання моніторингу промислового обладнання це такі речі, як двигуни та контролери двигуна. Для випадків використання Smart Environment це можуть бути датчики руху або зчитувачі значків. Для випадків використання відстеження активів - це GPS-трекери. Одним із викликів при виборі позаштатного обладнання може бути доступ до даних. Багато постачальників пропонують рішення, призначені для передачі даних, які можуть працювати добре для вирішення дуже конкретних проблем, але вони не працюють добре, коли ви хочете використовувати ці дані як частину більш широкого додатка IoT. Деякі постачальники можуть також мати хмарну службу, де ви можете отримати доступ до даних через API [17].

У більшості рішень IoT пристрої надсилають дані про стан та отримують команди з централізованої платформи IoT. Існує маса варіантів того, як здійснюється з'єднання між пристроєм і платформою, і це сильно залежить від середовища та обмежень самого пристрою. Якщо пристрій знаходиться на вулиці і рухається, як у випадках використання відстеження активів, мобільний зв’язок - це хороший вибір. Якщо пристрій знаходиться в приміщенні в домашніх або будівельних умовах, можливо, у вас є Ethernet або WiFi. Якщо пристрій працює від акумулятора, можливо, вам знадобиться вивчити параметри малої потужності, такі як Bluetooth з низьким рівнем енергії або LPWAN [18].

Деякі пристрої не можуть підключитися безпосередньо до центральної платформи та вимагають використання шлюзу IoT для усунення розриву між вашим локальним середовищем та вашою платформою. Це часто зустрічається в промислових умовах, де ви можете взаємодіяти з існуючим обладнанням через локальні протоколи, такі як Modbus, OPC UA або Serial. Шлюзи також потрібні при використанні бездротових технологій, таких як BLE та LPWAN, оскільки вони не забезпечують прямого підключення до вашої мережі чи хмари. У цих ситуаціях пристрій підключається до шлюзу. Шлюз зчитує необхідну інформацію, а потім надсилає дані на платформу, використовуючи підключення "backhaul", наприклад, WiFi, який може отримати доступ до вашої мережі або хмари [19].

Ваша платформа IoT - це центральний сховище даних та механізм оркестрації для вашого рішення.

Загалом, елемент інтернету речей можна поділити на чотири категорії. Це сервіси (Services), апаратна частина (Hardware), набір правил (Rules) та програмна частина (Software). Ці категорії також можна розділити на підгрупи за призначенням[20]

Архітектура IoT може відрізнятись в залежності від призначення та реалізації. Та загалом вона схожа на типову АСУТП (Автоматизована Система Управління Технологічними Процесами).

Взаємодія з середовищем відбувається через датчики (sensors) та виконавчі механізми, аналогічно, як це робиться в АСУТП для будь-якого об’єкту керування. Ці датчики разом з усією інфраструктурою для інтеграції з рівнем обробки подій через мережу Internet формують так звану граничну область[21].

Події, котрі надходять із граничної області, зберігають і опрацьовують відповідно до задачі. Дані що поступають з граничної області зберігаються і обробляються відповідно до задачі. На цьому рівні дані зберігають, опрацьовують, надсилають потрібним додаткам (Real-Time Message Brokering, Stream Processing). Додатково на цьому рівні здійснюють адміністрування та керування пристроями з граничної області. Отримання результатів, контроль, віддалене керування та адміністрування системи здійснюють через кінцеві застосунки з використанням

мережі Internet. Усі функції обробки, в тому числі аналітичні, котрі взаємодіють з сервісами керування даних через API (Application Program Interface)[22].

2.2 Протоколи і формати обміну даними між компонентами системи

Опишемо протоколи передачі даних між компонентами системи.

Зв’язок між пристроями (або шлюзами) і платформою відбувається по зашифрованому протоколу MQTT. IoT кидає виклик підприємствам, малим компаніям та розробники з новими проблемами для вирішення. Хоча HTTP є фактичним протоколом для людської мережі, комунікація між машинами в масштабі вимагає зміни парадигми, відхиляючись від запиту / відповіді та ведучого до публікації / підписки. Ось де ультралегкий, масово масштабований та простий у здійсненні протокол MQTT входить у зображення[23].

MQTT - це бінарний клієнт-сервер для підписки транспортного протоколу обміну повідомленнями, вперше стандартизованого OASIS. Він легкий, відкритий, простий і простий у виконанні. Розроблений з мінімальними накладними витратами, цей протокол є хорошим вибором для різноманітних програм Machine-to-Machine (M2M) та промислового Інтернету речей, особливо там, де потрібен невеликий слід коду. MQTT використовує багато характеристик транспорту TCP, тому мінімальна вимога для використання MQTT є робочий стек TCP, який тепер доступний навіть для найменших вбудованих систем[24].

У якості основного формату даних, у якому фронтальні сервери отримують інформацію, використовується JSON. JSON - це не просто проста заміна XML. Наразі JSON використовують для створення справді елегантних, корисних та ефективних додатків[24].

2.3 Мікросервісна архітектура

Архітектура мікросервісів – це методологія розробки програмного забезпечення для побудови додатків – варіант сервісно-орієнтованого архітектурного (SOA) стилю, який структурує додаток як сукупність слабко пов'язаних служб, які [25]:

- Нещільно з'єднані
- Незалежно розгортаються
- Організовано навколо можливостей бізнесу
- Володіють невеликими командами

Ця методика є принциповим зрушенням у підході IT до розробки ПЗ. Ідея полягає в тому, щоб розділити єдину монолітну програму на набір менших взаємопов’язаних сервісів. Перевага використання мікросервісів порівняно з більш традиційним підходом полягає в тому, що команди розробників здатні швидко створювати нові компоненти додатків, не впливаючи негативно на інші сервіси. Зміни однієї команди не зламають та не пошкодять весь додаток або його частини. Це допомагає охопити технологію DevOps і зробити доставку (CI / CD) більш безшовною рисунок 2.1.

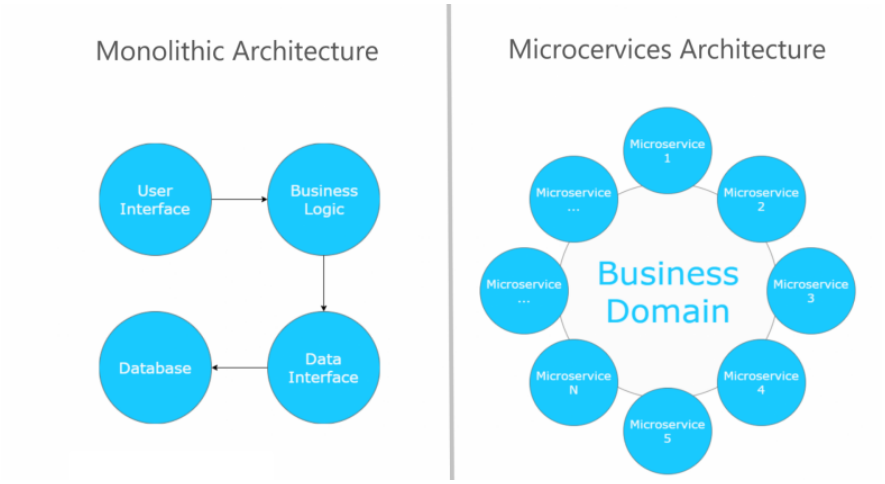


Рисунок 2.1 – Монолітна та мікросервісна архітектури

В останні роки ця тенденція набула популярності, коли підприємства прагнуть стати більш спритними та рухатися до CI та постійного тестування.

Архітектура мікросервісів дозволяє безперервно поставляти та розгортати великі, складні програми. Це також дозволяє організації розвивати свій стек технологій, масштабуватися та бути більш стійкими з часом. Ці блоки також забезпечують безперервну доставку та розгортання великих, монолітних додатків з мінімальною потребою в централізації. Особливості мікросервісів [27]:

- Спритність - будь-яку нову функцію можна швидко розробити та відкинути знову.
- Еластичність - завдяки розповсюдженню функціональності на декілька сервісів виключається сприйнятливість до однієї точки відмови.
- Рентабельність - більш швидкі повторення та скорочення часу простою можуть збільшити дохід.
- Гнучкість - різні мови та технології можна використовувати для створення різних служб одного і того ж додатка.
- Розв'язка - всі сервіси можна легко побудувати та масштабувати.
- Незалежність - команди можуть працювати незалежно один від одного
- Постійна доставка - дозволяє часто випускати програмне забезпечення та оновлення.
- Відповідальність - ставляться до кожної програми як до окремої, самодостатньої одиниці.
- Додаток на основі мікросервісів - це розподілена система, що працює на декількох процесах або послугах, як правило, навіть на декількох серверах або хостах. Кожен екземпляр служби, як правило, є процесом. Тому служби повинні взаємодіяти, використовуючи протокол зв'язку між процесами, такий як HTTP, AMQP або двійковий протокол типу TCP, залежно від характеру кожної послуги [28].

Спільнота мікросервісів пропагує філософію "розумних кінцевих точок". Цей слоган заохочує дизайн, який є максимально роз'єднаним між мікросервісами та максимально згуртованим у межах однієї мікросервіси. Як було пояснено раніше, кожна мікросервіс володіє власними даними та власною логікою домену[29].

Два протоколи, що часто використовуються, - це запит / відповідь HTTP з API-ресурсами ресурсів (найбільше запитів) та легкий асинхронний обмін повідомленнями під час передачі оновлень у декількох мікросервісах.

Клієнт і сервіси можуть спілкуватися через безліч різних типів зв'язку, кожен з яких орієнтований на різний сценарій та цілі. Спочатку такі типи комунікацій можна класифікувати за двома осями [29].

Перша вісь визначає, чи є протокол синхронним чи асинхронним:

- Синхронний протокол. HTTP - це синхронний протокол. Клієнт відправляє запит і чекає відповіді від служби. Це незалежно від виконання коду клієнта, яке може бути синхронним (потік заблоковано) або асинхронним (потік не заблокований, і відповідь зрештою надійде до зворотного дзвінка). Тут важливим моментом є те, що протокол (HTTP / HTTPS) є синхронним і клієнтський код може продовжувати своє завдання лише тоді, коли він отримає відповідь HTTP-сервера.
- Асинхронний протокол – інші протоколи, такі як AMQP (протокол, підтримуваний багатьма операційними системами та хмарними середовищами), використовують асинхронні повідомлення. Клієнтський код або відправник повідомлення зазвичай не чекають відповіді. Він просто надсилає повідомлення, як при надсиланні повідомлення в чергу RabbitMQ або будь-якого іншого брокера повідомлень.

Друга вісь визначає, чи має зв'язок один приймач або кілька приймачів [30]:

Один приймач. Кожен запит повинен бути оброблений точно одним приймачем або службою. Прикладом такого спілкування є шаблон команди.

Кілька приймачів. Кожен запит може бути оброблений від нуля до декількох приймачів. Цей тип зв'язку повинен бути асинхронним. Прикладом є механізм публікації / підписки, використовуваний у таких моделях, як архітектура, що керується подією. Зазвичай він реалізується через службову шину або подібний артефакт, як Azure Service Bus, використовуючи теми та підписки [31].

Додаток на основі мікросервісу часто використовуватиме поєднання цих стилів спілкування. Найпоширеніший тип - це зв'язок одного приймача із синхронним протоколом, як HTTP / HTTPS, коли викликається звичайна послуга

HTTP Web API. Мікросервіси також зазвичай використовують протоколи обміну повідомленнями для асинхронного зв'язку між мікросервісами [32].

Ні асинхронний характер виконання потоку клієнта, ні асинхронність вибраного протоколу не є важливими моментами інтеграції мікропослуг. Важливо, це можливість асинхронно інтегрувати свої мікросервіси, зберігаючи незалежність мікросервісів [32].

2.4 Підсистема прийому операційних даних

Кожна програма створює дані, будь то повідомлення журналу, метрики, активність користувача, вихідні повідомлення чи щось інше. Кожен байт даних має розповісти щось важливе, що сповістить про наступне, що потрібно зробити. Для того, щоб знати, що це таке, потрібно отримати дані з того місця, де вони створені, і де їх можна проаналізувати [33].

Чим швидше ми можемо це зробити, тим більш спритними та чуйними можуть бути наші організації. Чим менше зусиль ми витрачаємо на переміщення даних, тим більше ми можемо зосередитись на основному бізнесі. Ось чому підсистема прийому операційних даних є важливою складовою підприємства, керованого даними. Наше переміщення даних стає майже таким же важливим, як і самі дані.

Apache Kafka (Kafka) - це відкрита, розподілена потокова платформа, яка дозволяє розробляти додатки, керовані подіями в реальному часі [34].

Сьогодні мільярди джерел даних постійно генерують потоки записів даних, включаючи потоки подій. Подія - це цифровий запис дії, яка сталася, та часу, коли вона сталася. Зазвичай подія - це дія, яка рухає іншу дію як частину процесу. Клієнт, який замовляє замовлення, вибирає місце в рейсі або подає реєстраційну форму - все це приклади подій. Подія не повинна залучати людину - наприклад, підключений звіт термостата про температуру в даний момент часу також є подією.

Ці потоки пропонують можливості для додатків, які реагують на дані чи події в режимі реального часу. Платформа потокової передачі дозволяє розробникам створювати додатки, які постійно споживають та обробляють ці потоки з

надзвичайно високою швидкістю, з високим рівнем вірності та точності на основі правильного порядку їх виникнення [35].

LinkedIn розробила Kafka в 2011 році як високопрофесійний брокер повідомлень для власного користування, а потім відкрита і пожертвувала Kafka програмному фонду Apache Software. Сьогодні Kafka перетворилася на найбільш широко використовувану потокову платформу, здатну приймати та обробляти трильйони записів на день без будь-якого відчутного відставання в масштабі обсягів. 500 організацій Fortune 500, такі як Target, Microsoft, AirBnB і Netflix, покладаються на Kafka, щоб надавати своїм клієнтам досвід, керований даними, в реальному часі [35].

Kafka поєднує в собі дві моделі обміну повідомленнями, в черзі та підписки на публікацію, щоб забезпечити споживачам ключові переваги. Черга дозволяє поширювати обробку даних між багатьма споживчими інстанціями, що робить її високомасштабною. Однак традиційні черги - це не багато користувачів, які підписалися. Підхід публікувати-підписатися є багатоабонентським, але, оскільки кожне повідомлення надходить до кожного абонента, його не можна використовувати для розподілу роботи по декількох робочих процесах. Kafka використовує модель розділеного журналу, щоб зшити ці два рішення. Журнал - це впорядкована послідовність записів, і ці журнали розбиваються на сегменти або розділи, які відповідають різним абонентам. Це означає, що на одну і ту ж тему може бути декілька підписників, і кожному присвоюється розділ для забезпечення більшої масштабованості. Нарешті, модель Kafka забезпечує можливість відтворення, що дозволяє безлічі незалежних програм, зчитуваних з потоків даних, працювати самостійно за своєю швидкістю рисунок 2.2.

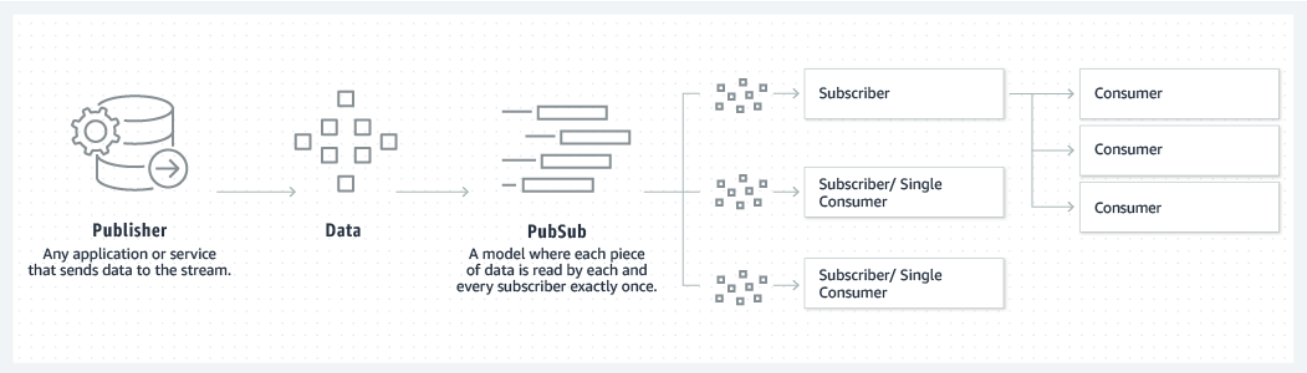


Рисунок 2.2 – Модель публікація - підписка

Переваги підходу Кафки:

- Масштабованість – розділена модель журналу Kafka дозволяє розподіляти дані на декількох серверах, що робить її масштабованою понад те, що підходило б на одному сервері.

- Швидкість – кафка відключає потоки даних, тому затримка дуже низька, що робить її надзвичайно швидкою.

- Міць – розділи розподіляються та реплікуються на багатьох серверах, а всі дані записуються на диск. Це допомагає захистити від збою сервера, роблячи дані дуже стійкими до відмов і довговічними.

Kafka – це розподілена платформа - вона працює як відмовлений, високодоступний кластер, який може охоплювати декілька серверів і навіть декілька центрів обробки даних. Теми Kafka розподіляються та реплікуються таким чином, щоб вони могли масштабувати, щоб обслуговувати великі обсяги одночасних споживачів, не впливаючи на продуктивність. Як результат, як повідомляє Apache.org, "Kafka виконає те саме, чи є у вас 50KB або 50TB постійного сховища на сервері" [35].

2.5 Підсистема обробки даних

Лямбда-архітектура – це спосіб обробки величезних кількостей даних (тобто "Big Data"), що забезпечує доступ до методів пакетної обробки та потокової обробки за допомогою гібридного підходу. Лямбда-архітектура використовується

для вирішення проблеми обчислення довільних функцій. Сама архітектура лямбда складається з 3 шарів рисунок 2.3.

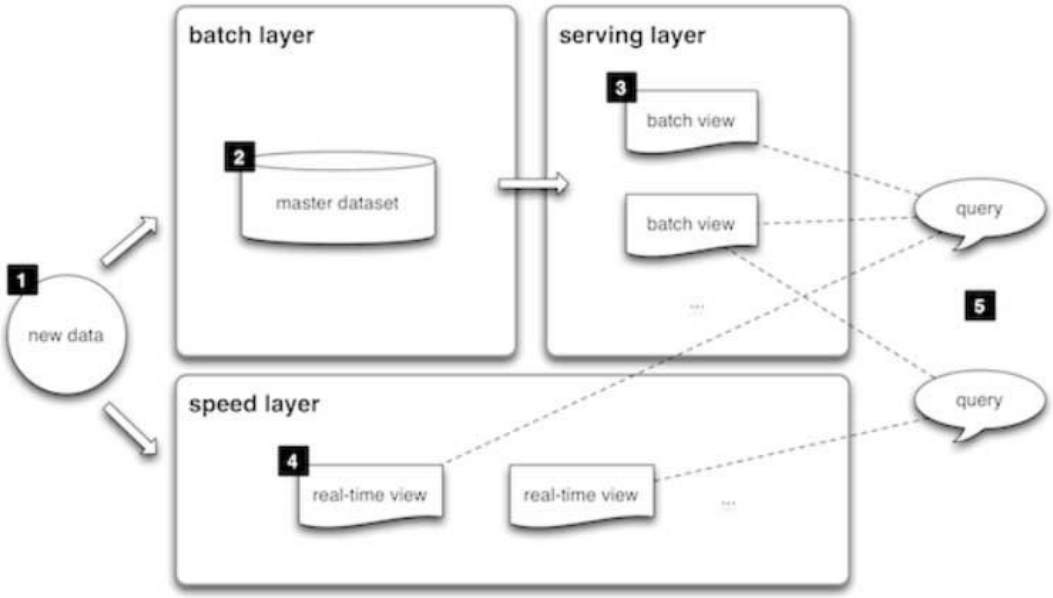


Рисунок 2.3 – Лямбда – архітектура

Пакетний шар – нові дані надходять постійно, як подача в систему даних. Він подається одночасно на пакетний шар і швидкісний шар. Він оглядає всі дані одразу і врешті-решт виправляє дані в потоковому шарі. Тут можна знайти багато ETL та традиційне сховище даних. Пакетний шар має дві дуже важливі функції [36]:

- Для управління основним набором даних.
- Попередньо обчислити перегляди партії.
- Обслуговуючий шар.

Виходи з пакетного шару у вигляді переглядів партії та ті, які надходять із шару швидкості у вигляді подань майже в режимі реального часу, передаються на сервісне обслуговування. Цей шар індексує перегляди пакетів, щоб їх можна було запитувати з низькою затримкою на спеціальній основі [37].

- Швидкісний шар (потоковий шар) – цей шар обробляє дані, які вже не доставлені в пакетному поданні через затримку пакетного шару. Крім того, він

стосується лише останніх даних для того, щоб забезпечити повний перегляд даних користувачеві, створюючи представлення в режимі реального часу.

Переваги лямбда-архітектури:

- Без управління сервером – вам не потрібно встановлювати, підтримувати або адмініструвати будь-яке програмне забезпечення.

- Гнучкі масштабування – ваша програма може бути автоматично масштабована або масштабована, регулюючи її ємність

- Автоматизована висока доступність – означає те, що безсерверні програми вже мають вбудовану доступність та толерантність до помилок. Це гарантує, що всі запити отримають відповідь про те, успішно чи ні.

- Бізнес-спритність – реагуйте в режимі реального часу на зміни сценаріїв бізнесу та ринку

Опишемо також підхід MapReduce, який частково застосовується для обробки даних платформою.

MapReduce – це процес складання списку об'єктів і виконання операції над кожним об’єктом у списку (тобто, карта), щоб або створити новий список, або обчислити одне значення (тобто зменшити) рисунок 2.4 [38].

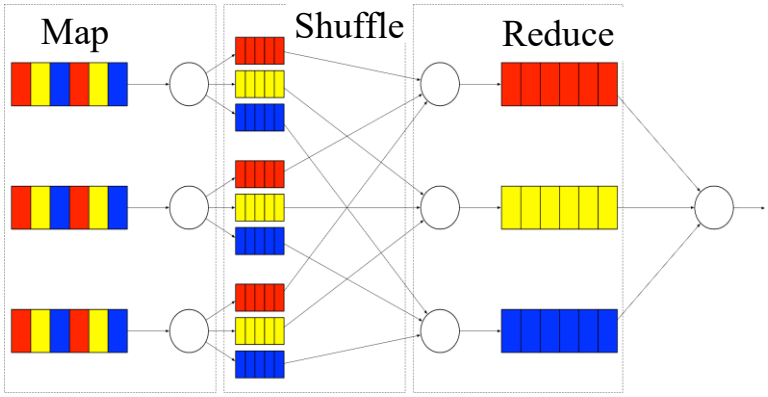


Рисунок 2.4 – Підхід MapReduce

Технічно паралельна обробка стосується використання декількох машин, які вносять свої ядра оперативної пам'яті та процесора для обробки даних. Це концепція Hadoop, де ви не тільки зберігаєте дані на різних машинах, але також

можете обробляти дані локально [39].

Відображення (Map) і зменшення (Reduce):

- Клас Map обробляє етап відображення; він відображає дані, присутні в різних вузлах даних.
- Клас Reduce обробляє редукційну фазу; він агрегує і зменшує вихід різних вузлів даних для отримання кінцевого результату.

Дані, що зберігаються на декількох машинах, проходять через Map. Кінцевий вихід отримується після перемішування, сортування та зменшення даних

Вхідні розбиття:

- Вхід до завдання MapReduce ділиться на шматки фіксованого розміру, які називаються вхідними розколами
- Картографування – це найперший етап у виконанні програми зменшення карт. У цій фазі дані в кожному розділі передаються функції відображення для отримання вихідних значень. У нашому прикладі завдання етапу відображення полягає у підрахунку кількості входів кожного слова з вхідних розділів (детальніше про введення-розбиття наведено нижче) та підготувати список у вигляді слово - частота.
- Перемішування – ця фаза споживає вихід фази картографування. Її завдання - консолідувати відповідні записи з виведення фази Mapping. У нашому прикладі одні й ті ж слова поєднуються разом із відповідною частотою.
- Зменшення – у цій фазі вихідні значення з фази перетасовування агрегуються. Ця фаза поєднує значення з фази переміщення та повертає єдине вихідне значення. Коротше кажучи, ця фаза підсумовує повний набір даних.

Вимоги, яким має відповідати підсистема обробки даних[39]:

- Висока продуктивність. Продуктивність потокової обробки не має бути вузьким місцем у системі. Аналітичні запити також потрібно виконувати якомога швидше.
- Масштабованість. Можливість підвищити пропускну здатність ланцюжка потокової обробки даних чи прискорити аналітичні запити за допомогою збільшення кількості worker-серверів.

- Наявність інтерфейсів зв’язку із базами даних і чергами повідомлень. Підтримка таких API на рівні системи дозволяє спростити розробку чи адаптацію під змінені джерела чи сховища даних.
- Бібліотека операцій і алгоритмів. Як мінімум, система повинна підтримувати базові операції статистичного аналізу.
- Можливість розширення функціональності бібліотеки. Система має бути розширюваною, аби за необхідності мати можливість додати якийсь специфічний крок обробки даних чи алгоритм.

2.5.1 Apache Spark – система обробки даних

Для задоволення вищеозначених потреб оберемо систему Apache Spark[39].

Опишемо детальніше її можливості та переваги.

Apache Spark – це швидка кластерна обчислювальна система загального призначення. Він надає API високого рівня в Java, Scala, Python та R, а також оптимізований движок, який підтримує загальні графіки виконання. Він також підтримує багатий набір інструментів вищого рівня, включаючи Spark SQL для SQL та структуровану обробку даних, MLlib для машинного навчання, GraphX для обробки графіків та Spark Streaming. Розроблена у 2009 році у AMPLab (UC Berkeley), публічний реліз як Apache проекту відбувся у 2010 році [40].

Ядро Spark доповнюється набором потужних бібліотек вищого рівня, які можна легко використовувати в одному додатку. Наразі ці бібліотеки включають SparkSQL, Spark Streaming, MLlib (для машинного навчання) та GraphX, кожна з яких детальніше детальніше описана в цій статті. В даний час розробляються додаткові бібліотеки і розширення Spark.

Spark має значні переваги порівняно з іншими Big Data і MapReduce технологіями, такими як Hadoop і Storm. Apache Spark досягає високої продуктивності як для пакетних, так і потокових даних, використовуючи сучасний DAG-планувальник, оптимізатор запитів і механізм фізичного виконання. Spark пропонує понад 80 операторів високого рівня, які спрощують створення

паралельних додатків. Його можна використовувати інтерактивно з оболонок Scala, Python, R та SQL. Оболонка Spark забезпечує простий спосіб вивчення API, а також потужний інструмент для інтерактивного аналізу даних. Він доступний або в Scala (який працює на Java VM і тому є хорошим способом використання існуючих бібліотек Java). [41].

Spark Core - це базовий двигун для масштабної паралельної та розподіленої обробки даних. Він відповідає за управління пам’яттю та відновлення несправностей, планування, розповсюдження та моніторинг завдань на кластері.

Spark використовує концепцію RDD (Resilient Distributed Dataset), незмінного відмови, розподіленого набору об'єктів, якими можна керувати паралельно. RDD може містити будь-який тип об'єкта і створюється завантаженням зовнішнього набору даних або розповсюдженням колекції з драйверної програми.

RDD підтримують два типи операцій:

- Перетворення - це операції (такі як карта, фільтр, з'єднання, об'єднання тощо), які виконуються на RDD і дають новий RDD, що містить результат.
- Дії - це операції (такі як зменшення, підрахунок, спочатку тощо), які повертають значення після виконання обчислення на RDD.

Трансформації в Spark "ліниві", тобто вони не підраховують свої результати відразу. Натомість вони просто «запам’ятовують» операцію, яку потрібно виконати, і набір даних (наприклад, файл), для якого слід виконати операцію. Перетворення фактично обчислюються лише тоді, коли викликається дія і результат повертається до драйверної програми. Така конструкція дозволяє Spark працювати більш ефективно. Наприклад, якщо великий файл був перетворений різними способами і перейшов до першої дії, Spark буде лише обробляти і повертати результат для першого рядка, а не виконувати роботу для всього файлу.

За замовчуванням кожен перетворений RDD може бути перерахований щоразу, коли виконується дія на ньому. Однак ви також можете зберегти RDD в пам'яті за допомогою методу persist або кешу, і в цьому випадку Spark збереже елементи навколо кластеру для набагато швидшого доступу наступного разу, коли ви його запитуєте [38].

Spark Streaming підтримує обробку потокових даних у режимі реального часу, таких як файли журналів виробничого веб-сервера (наприклад, Apache Flume та HDFS / S3), соціальних медіа, таких як Twitter, та різних черг для обміну повідомленнями, таких як Kafka. В середені, Spark Streaming отримує вхідні потоки даних і ділить їх на групи. Далі вони обробляються двигуном Spark і генерують остаточний потік результатів партіями рисунок 2.5.

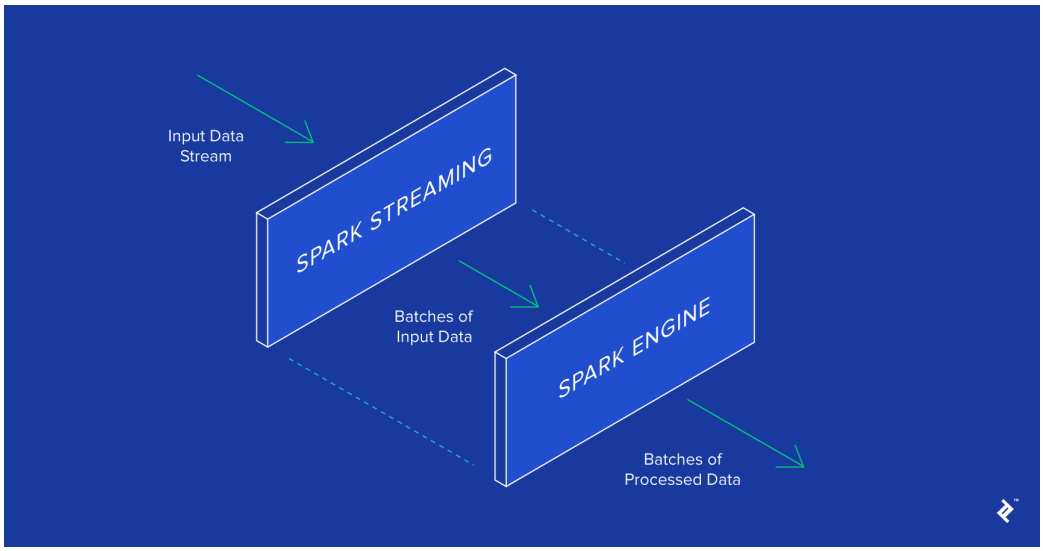


Рисунок 2.5 – Генерування результатів Spark

API Spark Streaming тісно відповідає інтерфейсу Spark Core, полегшуючи роботу програмістів у світі як пакетних, так і потокових даних.

2.6 Підсистема зберігання операційних даних

MongoDB - орієнтована на документи база даних NoSQL, яка використовується для зберігання даних з великим обсягом. Замість використання таблиць та рядків, як у традиційних реляційних базах даних, MongoDB використовує колекції та документи. Документи складаються з пар ключових значень, які є базовою одиницею даних у MongoDB. Колекції містять набори документів та функції, що є еквівалентом таблиць реляційних баз даних. MongoDB – це база даних, яка з'явилася на світ приблизно в середині 2000-х.

Кожна база даних містить колекції, які в свою чергу містять документи. Кожен документ може бути різним із різною кількістю полів. Розмір і зміст кожного документа можуть відрізнятися один від одного. Структура документа більше відповідає тому, як розробники конструюють свої класи та об’єкти у відповідних мовах програмування. У рядках (або документах, як їх називають у MongoDB) не потрібно заздалегідь мати схему. Натомість поля можна створювати на льоту. Модель даних, доступна в MongoDB, дозволяє представити ієрархічні відносини, легше зберігати масиви та інші більш складні структури. Середовища MongoDB дуже масштабовані. Компанії по всьому світу визначили кластери, де деякі з них працюють 100 + вузлів із мільйонами документів у базі даних.

Сьогодні переваги баз даних NoSQL не є секретом, особливо коли хмарні обчислення набули широкого поширення [40].

Бази даних NoSQL були створені у відповідь на обмеження традиційної технології реляційних баз даних. У порівнянні з реляційними базами даних, бази даних NoSQL є більш масштабованими та забезпечують високу продуктивність, а їх модель даних усуває ряд недоліків реляційної моделі.

До переваг бази даних NoSQL, таких як MongoDB, можна віднести:

- Великі обсяги структурованих, напівструктурованих та неструктурованих даних.
- Об'єктно – орієнтоване програмування, яке просте у використанні та гнучко.
- Ефективна масштабна архітектура замість дорогої монолітної архітектури.

Сьогодні компанії використовують бази даних NoSQL для збільшення кількості випадків використання. Бази даних NoSQL також мають тенденцію бути відкритим кодом, що означає відносно недорогий спосіб розробки, впровадження та спільного використання програмного забезпечення.

- Універсальність – оскільки однією з визначальних особливостей MongoDB є база даних NoSQL (неструктурована мова запитів), це її структура без даних або нереляційних даних. Це дозволяє забезпечити величезну ступінь універсальності для зберігання різних типів даних та доступу до них на льоту. Універсальність особливо важлива в даний час з комодитизацією великих даних, яка генерується з

незліченних різних джерел і не завжди належить до акуратних категорій.

- Горизонтальне масштабування – ще одна перевага, яку пропонує MongoDB, - це можливість горизонтального масштабування шляхом заточування. Оскільки збережені дані не структуровані вертикально, вони можуть бути розповсюджені або «пошаровані» на декількох серверах, з можливістю легко додавати більше за необхідності. Крім того, заточування полегшує апаратну сторону речей, оскільки полегшує необхідну потужність зберігання та обробки для однієї машини.

- Швидкість - пов'язана з горизонтальним масштабуванням - швидкість MongoDB. Його робочий процес для подання ключів запиту простіший, ніж у SQL, оскільки для нього не потрібно вказувати схему - просто індексуйте точку даних, яку ви шукаєте, і MongoDB отримає її. Відсутність заданої реляційної структури означає, що для подання запиту потрібно значно менший обробний потенціал для пошуку та отримання, ніж у реляційній базі даних.

- Доступність – MongoDB підтримує всі основні мови програмування (Ruby, PHP, Java тощо), а також має численні драйвери, що підтримуються спільнотою, і для маловідомих мов програмування. Він може розміщувати власний хмарний сервіс Atlas і пропонує як керований спільнотою відкритий код, так і преміальне корпоративне видання - наш GUI та IDE для MongoDB, Studio 3T, працює з будь-яким із цих розгортань.

- Гнучкість – як ви вже здогадалися, нереляційне горизонтальне масштабування MongoDB дозволяє досягти величезної міри операційної гнучкості. Це робить його корисною платформою для експериментів з новими нетрадиційними моделями контенту. Це також робить його безцінним для тих, хто змінює вимоги до вмісту, наприклад рекламних оголошень. Однією з найважливіших причин того, що MongoDB настільки широко прийнятий, є простота, з якою він може інтегруватися в конвеєр розвитку - розробникам не потрібно вивчати SQL або наймати адміністратора бази даних, щоб повною мірою використовувати його функціонал.

- Транзакції – MongoDB підтримує багатодокументні ACID (атомні / послідовні / ізольовані / довговічні) транзакції. Операції гарантують, що передача даних відбувається або успішно, або взагалі не відбувається. У минулому

банки та інші великі організації з обережністю використовували MongoDB через відсутність транзакційної цілісності. Операції є основною причиною того, що MongoDB швидко перетнув прірву від нішевого програмного забезпечення до руйнуючої на ринку платформи основної бази даних.

2.6.1 База даних Apache Cassandra

Розглянемо рішення для зберігання даних для завантаження події IoT. З'являються кілька питань: Як система зберігає всі дані зі змінною довжиною події? І як система запитує масивний, швидко зростаючий набір даних для негайного розуміння та ітеративних, постійних удосконалень.

Ці речі потребують розподіленого сховища даних, який може вмістити записи, що розвиваються, і змінної довжини з великим масштабом і швидкістю прийому, використовуючи вбудовані відмовостійкість і доступність з високою швидкістю запису та гідною швидкістю читання. І ці дані повинні бути керовані мовою запитів [36].

У сучасну інформаційну епоху мільярди підключених пристроїв та цифрових середовищ постійно передають та зберігають дані. Від смартфонів та ноутбуків, веб – браузерів та додатків, до розумних приладів, інфраструктурного контролю та датчиків – усі ці пристрої генерують дані.

Кожен шматочок генерованих даних створюється для збору, зберігання, вдосконалення, запиту, аналізу та експлуатації з метою постійного вдосконалення: постійно та ітеративно забезпечуючи кращі, безпечніші та ефективніші продукти, процеси та послуги.

Генерування даних нескінченне, і ці дані, зберігаючись, зростають експоненціально. Поки користувачі продовжують використовувати цифрові продукти, і поки цифрові продукти залишаються підключеними до мереж, вони продовжуватимуть це робити.

Apache Cassandra – це NoSQL з відкритим кодом, який може швидко обробляти та обробляти величезну кількість даних. Це також децентралізоване,

розподілене, масштабоване, високодоступне, стійке до відмов та налаштоване послідовно, з однаковими вузлами, згрупованими разом, щоб усунути окремі точки відмови та вузькі місця [36].

Модель даних Кассандри складається з стовпців, рядків, сімейств стовпців та простору ключів. Давайте розглянемо кожну частину детально.

Стовпець - найосновніша одиниця в моделі даних Кассандри, і кожен стовпець складається з імені, значення та часової позначки.

Рядок - сукупність стовпців, позначених назвою.

У таблиці 2.1 показано структуру даних; ключ рядка (row key) виконує роль ідентифікатора колонки, яка слідує за ним, а назва колонки та значення зберігаються у суміжних блоках.

Таблиця 2.1 – Структура даних у Cassandra

Ключ рядка	Колонка 1	…	Колонка N
	Значення 1	…	Значення N
	Сімейство колонок		

Розглянемо приклад зберігання інформації для сценарію роботи з метеорологічними даними. У таблиці 2.2 ключем рядка є ідентифікатор пристрою (“deviceId”), а в колонках “temperature” і “pressure” записані відповідно значення температури та тиску[39].

Варто зауважити, що в кожному рядку зберігаються не тільки значення, а і назви колонок, що дозволяє структурі (schema) бути динамічною.

Кассандра складається з безлічі вузлів зберігання та зберігає кожен рядок у межах одного вузла зберігання. У межах кожного ряду Cassandra завжди зберігає стовпці, відсортовані за їх назвами. Використовуючи такий порядок сортування, Cassandra підтримує запити фрагментів, де дано рядок, користувачі можуть отримати підмножину його стовпців, що входять до заданого діапазону назв стовпців.

Таблиця 2.2 – Приклад зберігання даних

deviceId = 1	temperature	pressure
	24.3	738.4
deviceId = 2	temperature	pressure
	16.1	762.9

Keyspace – група з багатьох сімей колонок разом. Це лише логічне групування сімейств стовпців і забезпечує ізольоване поле для імен.

Відмінності між RDBMS та Cassandra:

- Не приєднується – не може виконувати приєднання до Кассандри. Якщо моделі даних потрібно щось на зразок з'єднання, вона повинна або виконати роботу на стороні клієнта, або створити денормалізовану другу таблицю, яка представляє результати об'єднання. Цей останній варіант є кращим при моделюванні даних Кассандри.

- Немає референтної цілісності – хоча Кассандра підтримує такі функції, як легкі транзакції та партії, сама Кассандра не має поняття референтної цілісності в таблицях. У реляційній базі даних можна вказати сторонні ключі в таблиці для посилання на первинний ключ запису в іншій таблиці. Але Кассандра цього не виконує. Досі існує загальна вимога до дизайну для зберігання ідентифікаторів, пов'язаних з іншими об'єктами, у таблицях, але такі операції, як каскадні делети, недоступні [26].

- Денормалізація – у дизайні реляційних баз даних часто потрібне значення нормалізації. Це не є перевагою при роботі з Кассандрою, оскільки вона найкраще працює, коли модель даних денормалізована. Часто трапляється так, що компанії також закінчують денормалізацію даних у реляційних базах даних. Для цього є дві загальні причини. По - перше, це продуктивність.

Друга причина того, що реляційні бази даних денормалізуються за призначенням – це структура ділових документів, яка потребує збереження. Тобто

додаюча таблиця, яка посилається на безліч зовнішніх таблиць, дані яких можуть змінитися з часом, але потрібно зберегти додаючий документ як знімок історії.

У реляційному світі денормалізація порушує нормальні форми Кодда, і ви намагаєтесь цього уникнути. Але в Кассандрі денормалізація – це цілком нормально. Це не потрібно, якщо модель даних проста.

- Дизайн спочатку запиту – реляційне моделювання, простіше кажучи, означає, що починається з понятійного домену, а потім представляє іменники в домені в таблицях. Таблиці приєднання не існують у реальному світі і є необхідним побічним ефектом роботи реляційних моделей. Після викладених таблиць можна починати писати запити, які збирають різні дані, використовуючи зв'язки, визначені ключами. Запити у реляційному світі дуже вторинні. Передбачається, що завжди можна отримати потрібні дані, якщо ви правильно моделювали свої таблиці. Навіть якщо доводиться використовувати кілька складних підзапитів або об'єднати заяви, це зазвичай так.

2.7 Організація автентифікації. JSON Web Token

JSON Web Token (JWT) – це відкритий стандарт (RFC 7519), який визначає компактний та автономний спосіб безпечної передачі інформації між сторонами як об'єкт JSON. Цю інформацію можна перевірити і довірити, оскільки вона підписана цифровим шляхом. JWT можуть бути підписані за допомогою секретного (за допомогою алгоритму HMAC) або пари відкритого / приватного ключів за допомогою RSA або ECDSA [38].

Підписані маркери можуть перевірити цілісність претензій, що містяться в ньому, а зашифровані маркери приховують ці претензії від інших сторін. Коли маркери підписуються за допомогою пар відкритих / приватних ключів, підпис також засвідчує, що лише сторона, що тримає приватний ключ, є тією, яка його підписала.

Сценарії, в яких корисні веб-маркери JSON [27]:

- Авторизація: Це найпоширеніший сценарій використання JWT. Після входу користувача кожен наступний запит буде включати JWT, дозволяючи користувачеві отримувати доступ до маршрутів, служб та ресурсів, дозволених за допомогою цього маркера. Single Sign On – це функція, яка сьогодні широко використовується JWT, через її невеликі накладні витрати та її можливість легко використовуватись у різних областях.

- Обмін інформацією: веб-токени JSON – це хороший спосіб надійної передачі інформації між сторонами. Оскільки JWT можуть бути підписані (наприклад, за допомогою пар відкритих / приватних ключів), ви можете бути впевнені, що відправники такі, про які вони кажуть. Крім того, оскільки підпис обчислюється за допомогою заголовка та корисного навантаження, ви також можете переконатися, що вміст не був підроблений.

У своєму компактному вигляді веб-маркери JSON складаються з трьох частин, розділених крапками (.), які є: Заголовок, Корисний вантаж, Підпис. Тому JWT зазвичай виглядає наступним чином: xxxxx. yyyyy. zzzzz.

Під час аутентифікації, коли користувач успішно ввійде у систему, використовуючи свої облікові дані, буде повернутий веб-маркер JSON. Оскільки маркер є повноваженнями, необхідно дуже уважно ставитися до запобігання проблемам безпеки. Взагалі, ви не повинні тримати маркер довше, ніж потрібно.

Щоразу, коли користувач хоче отримати доступ до захищеного маршруту або ресурсу, агент користувача повинен надсилати JWT, як правило, у заголовку Авторизації за допомогою схеми Bearer. Зміст заголовка повинен виглядати наступним чином: авторизація – носій

У деяких випадках це може бути механізм без дозволу. Захищені маршрути сервера перевірять чи дійсний JWT у заголовку Авторизації, і якщо він присутній, користувачеві буде дозволено отримати доступ до захищених ресурсів. Якщо JWT містить необхідні дані, необхідність запиту бази даних щодо певних операцій може бути зменшена, хоча це не завжди може бути так.

Якщо маркер буде надісланий у заголовку авторизації, спільний розподіл ресурсів (CORS) не буде проблемою, оскільки він не використовує файли cookie.

Додаток або клієнт вимагає авторизації на сервері авторизації. Це виконується через один з різних потоків авторизації. Наприклад, типовий веб-додаток, сумісний з OpenID Connect, пройде через / oauth / autize кінцеву точку, використовуючи потік коду авторизації.

Коли авторизація надана, сервер авторизації повертає маркер доступу до програми. Додаток використовує маркер доступу для доступу до захищеного ресурсу (наприклад, API).

Зауважте, що з підписаними маркерами вся інформація, що міститься в токені, піддається користувачам або іншим сторонам, хоча вони не в змозі її змінити. Це означає, що не слід вносити секретну інформацію в маркер.

Переваги веб-токенів JSON (JWT) у порівнянні з простими маркерами веб-маркерів (SWT) та мовою розмітки мови твердження безпеки (SAML):

- Оскільки JSON менш багатослівний, ніж XML, при його кодуванні його розмір також менший, що робить JWT більш компактним, ніж SAML. Це робить JWT хорошим вибором для передачі у середовищах HTML та HTTP.
- З точки зору безпеки, SWT може бути симетрично підписаний спільним секретом за допомогою алгоритму HMAC. Однак маркери JWT та SAML можуть використовувати пара відкритих / приватних ключів у вигляді сертифіката X.509 для підписання. Підписання XML за допомогою цифрового підпису XML без введення незрозумілих отворів у безпеці дуже складно порівняно з простотою підписання JSON.
- JSON-парсери поширені в більшості мов програмування, оскільки вони відображають безпосередньо об'єкти. І навпаки, XML не має природного відображення документа-об'єкта. Це полегшує роботу з JWT.

2.8 Висновки до розділу 2

В даному розділі було відібрано сучасні системи, протоколи та бази даних. Використані сторонні рішеня отримали належнео обґрунтування. Також, відкритий код цх застосунків спрощує розширння системи.

Було детально описано різноманітні аспекти обраних архітектурних рішень: від обґрунтування високорівневого поділу на функціональні модулі до особливостей ефективної реалізації вводу/виводу та впливу обраної схеми автентифікації на масштабованість архітектури платформи.

Універсальність запропонованої системи та її архітектурне рішення робить можливим застосовування для побудови різних систем обробки даних з різних областей.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПЛАТФОРМИ

Практичну реалізацію було виконано мовою програмування Java (версія 8). На рисунку 3.1 наведено проект програмного продукту.

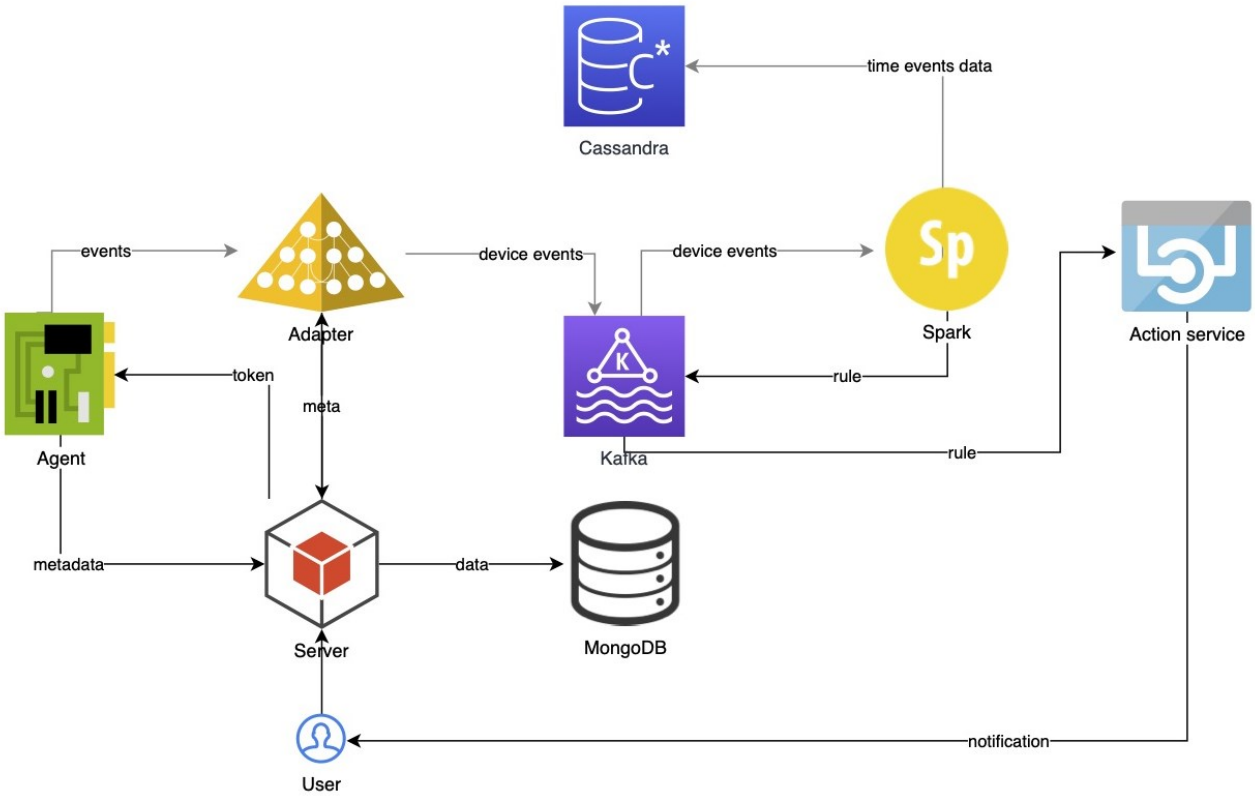


Рисунок 3.1 – Проект програмного продукту

У роботі були використанні наступні бібліотеки:

- Kafka Clients – клієнтські інтерфейси для взаємодії із кластером Apache Kafka.
- Набір бібліотек та інтерфейсів для роботи з Apache Spark.
- Spark-Cassandra Connector – драйвер Apache Cassandra з інтерфейсом, який інтегрований із API Apache Spark.
- Java mail library – бібліотека для відправлення електронного повідомлення.
- MongoDB Async Driver – драйвер MongoDB, реалізований з використанням Java NIO для неблокуючих запитів до бази даних.
- Бібліотека для створення та перевірки JSON Web Token.
- Apache Commons Codec – набір реалізацій кодеків, наприклад, Base64 для

формування JSON Web Token.

- SLF4J – Java API для логування різноманітних інформаційних і діагностичних повідомлень про хід роботи програми.
- Logback – реалізація SLF4J.
- JUnit – бібліотека для юніт-тестування

3.1 Agent – постачальник даних

Agent - це програмне забезпечення, яка встановлюється на пристрій. Його мета - збирання данних та відправка їх на платформу. Для цього використовується mqtt - протокол, клієнт якого вбудовується в Agent, а mqtt - топіки вказуються при налаштуванні пристрою. Також, для початку роботи, агенту необхідно зробити запит на отримання токену на Server. Для цього він використовує REST запит. Наразі, роль Agent виконує програма симулятор реалізована на Java.

Структура запиту на отримання токену:

- POST /api/ endpoints
- Запит
- Властивості: metadata: (object) - Метадані кінцевої точки складаються з декількох пар ключів і значень (називаються полями), де ключі є рядками, а значення мають будь-який тип JSON.

Приклад:

```
{
  "metadata": {
    "name": "sn-231",
    "year": 2020
  }
}
```

- Відповідь:
- Властивості: token: маркер кінцевої точки

Приклад:

```
{
  "token": "02226466-e744-48ac-8f0c-a57fe4e77de4"
}
```

3.1.2 Структура event для відправки по mqtt

topic: v1/srv/{token}/json/{requestId}

Приклад:

```
{
   "timestamp": 1586525376761,
   "data": {
      "temperature": "5",
      "weight": "6",
      "humidity": "2",
      "counter": "1",
      "speed": "4"
   }
}
```

3.2 Adapter - сервіс

Adapter - це сервіс, головна функція якого отримати по mqtt event від Agent та адаптувати його для подальшої обробки, збагатити метаданними та відправити в Kafka. Реалізован на Java.

Запит на отримання метадати із Server:

GET /api/ endpoints /{endpointId}/metadata

- Параметри: endpointId: Ідентифікатор кінцевої точки, який потрібно опрацювати.
- Відповідь

Приклад:


```
{
  "name": "sn-231",
  "year": 2020,
  "deviceModel": "MyDevice A300",
  "state": "REGISTERED"
}
```


Струкрутра event який потрапляє в Kafka

topic : events


```
{
  "endpointId": 55,
  "token": "02226466-e744-48ac-8f0c-a57fe4e77de4",
  "dataSample": {
    "data": {
```

```
      "temperature": "5",
      "weight": 6,
      "humidity": "2",
      "counter": "1",
      "speed": "4"
    },
    "timestamp": 1586525376761,
    "deviceMetadata": {
      "name": "sn-231",
      "year": 2020,
      "deviceModel": "MyDevice A300",
      "state": "REGISTERED"
    }
  }
}
```


3.4 Spark – обробка events

Сервіс в який інтегровано Spark - це елемент системи, який відповідає за обробку events. Він вичитує з kafka events топіку events та виявляє, чи справно працює дивайс, тобто, чи всі його показники є в нормі. Для цього сервіс звертається до Server для того, щоб отримати Rule - правило, за яким буде перевірятися event.

Запит на отримання rule :

GET /api/rules/{endpointId}

Відповідь

{
  "id": 15,
  "name": "speedCheck",
  "conditionQuery": "speed>5",
  "active": true,
  "applyToAllDevices": false,
  "devices": [
    "02226466-e744-48ac-8f0c-a57fe4e77de4"
  ]
}

Якщо rule тригереться, то Spark процессор пише event в Kafka в топік rule. Також, незалежно від rule всі events потравляють в Cassandra.

3.5 Action Service

Action Service - сервіс, головна задача якого вичитувати з Kafka rule топіку events, отримувати Action із Server та відправляти повідомлення Користувачеві. Реалізован на Java.

Запит на отримання action:

GET api/rules/{ruleId}/actions

Приклад

{
  "id": 27190,

"name": "test132",

"responseType": "SMS",

"responseJson": «{\"to\":\"1234567\",\"message\":\"Швидкість 02226466-e744-48ac-8f0c-a57fe4e77de4 більша за 5 м\с\»}»,

"ruleId": 15

}

3.6 Server

Server - це сервіс, із яким взаємодіє користувач. Має REST API якого описано у таблиці 3.1

Таблиця 3.1 – REST API

Метод	Адреса	Опис
1	2	3
POST	/api/auth /login	отримання токену для вхожу в систему
POST	/api/users	створення нового користувача
GET	/api/users	отримання всіх користувачів системи

Продовження таблиці 3.1

1	2	3
GET	/api/users/{ userId }	отримати користувача по ідентифікатору
DLETE	/api/users/{ userId }	видалити користувача по ідентифікатору
POST	/api/ endpoints	створення нового пристрої
GET	/api/ endpoints	отримати всі пристрої
GET	/api/ endpoints /{endpointId}	отримати пристрій по ідентифікатору
PUT	/api/ endpoints /{endpointId}	оновити інформацію про пристрій по ідентифікатору
DLETE	/api/ endpoints /{endpointId}	видалити пристрій по ідентифікатору
GET	/api/endpoints/{endpointId}/metadata	тримати метадату пристрою по ідентифікатору
POST	api/rules	створення нового правила
GET	api/rules	отримати всі правила

Продовження таблиці 3.1

1	2	3
GET	api/rules/{ruleId}	отримати правило по ідентифікатору
PUT	api/rules/{ruleId}	оновити інформацію про правило по ідентифікатору
DELETE	api/rules/{ruleId}	видалити правило по ідентифікатору
GET	api/rules/{endpointId}	отримати всі правила для пристрою
GET	api/rules/actions	отримати всі actions для правила

3.7 Взаємодія платформи з пристроями

Для демонстрації взаємодії платформи було використано Agent. Щоб він почав роботу необхідно було отримати токен від серверу. Спочатку необхідно увійти у систему серверу рисунок 3.2

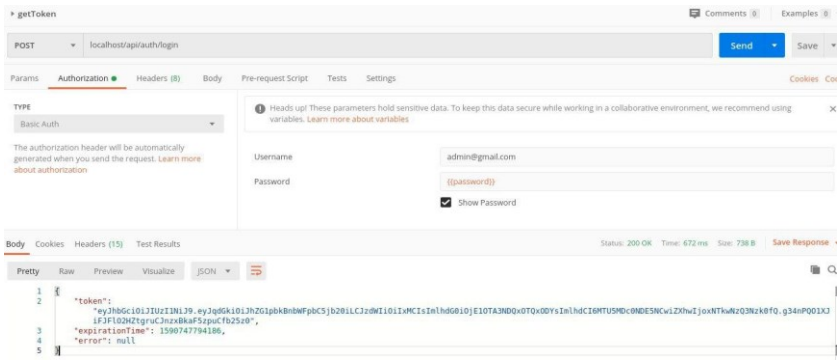


Рисунок 3.2 – Вхід у систему

За допомогою REST виклику POST /api/endpoints та отриманого токену з рисунку 3.2 створюємо новий дивайс в системі рисунок 3.3.

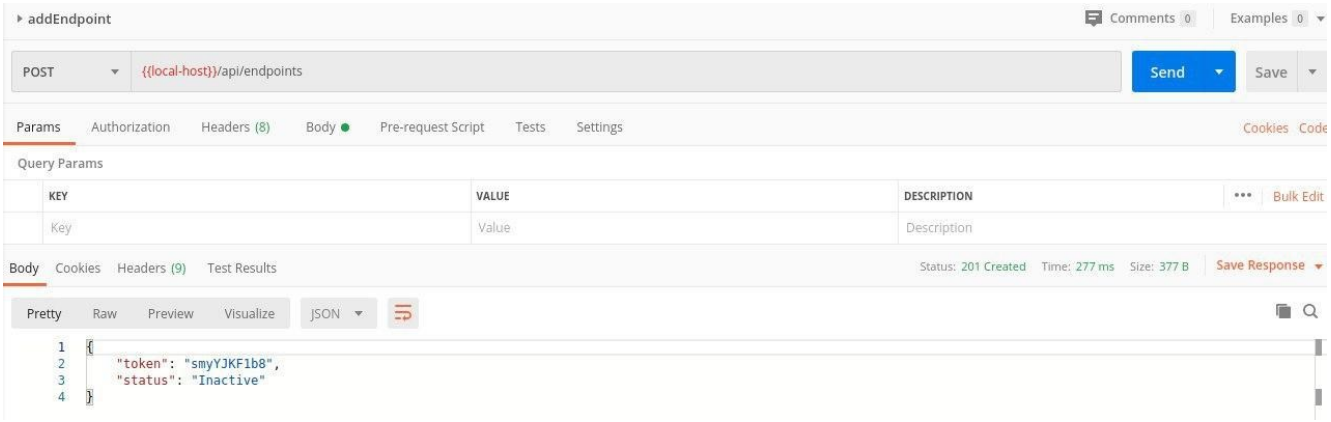


Рисунок 3.3 – створення нового дивайса

Із отриманим токеном запускаємо agent, він автоматично починає надсилати метрики рисунко 3.4.

Рисунок 3.4 – відправка даних

Дані з рисунку 3.4 потрапляють на Адаптер, який збагачує їх метаданими. Щоб їх створити необхідно зробити REST виклик на сервер рисунок 3.5.

Рисунок 3.5 – створення метадати

Наступним кроком Адаптер оброблені дані записує в Kafka Додаток А. Ці дані оброблює Спарк, зберігає їх, записує в Casandra (рис. 3.6) та Kafka в topic rule Додаток Б.

serial_number	tag	splitter	timestamp	partition_offset	value
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454823462	1-3462821	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454821457	1-3462820	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454821456	1-3462819	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454819450	1-3462818	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454819449	1-3462817	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454817444	1-3462816	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454817442	1-3462815	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454815437	1-3462814	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454815436	1-3462813	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454813431	1-3462810	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454813430	1-3462809	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454813429	1-3462808	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454809422	1-3462805	345
xl9N2qrzycQIWly_veXlF9ZIMq0	stamp	153	1589454809420	1-3462804	345

Рисунок 3.6 – дані в Casandra

Action service читає дані з Kafka topic rule та надсилає повідомлення користувачу рисунок 3.7.

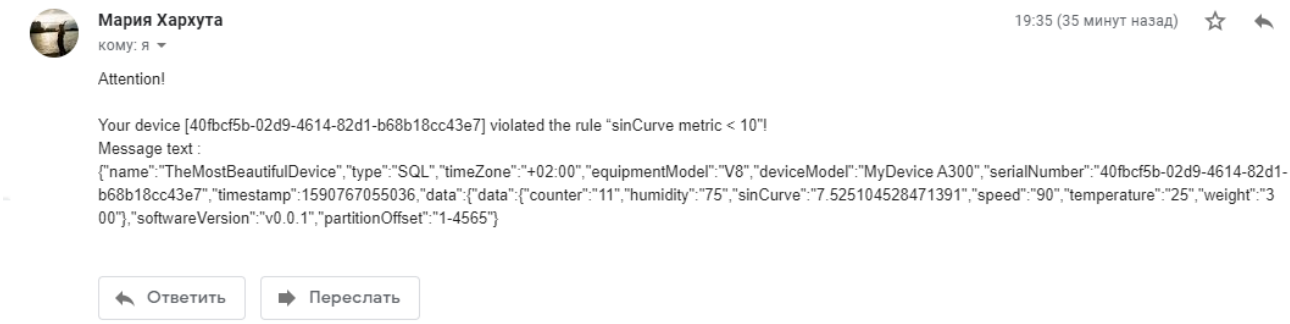


Рисунок 3.7 – повідомлення користувачу

3.8 Висновки до розділу 3

Запропоновано та реалізовано мікросервісну платформу, яка дозволяє ідентифікувати аварійну ситуацію на підприємстві.

Розроблене рішення є добре розширюваним у тому сенсі, що його можливо легко адаптувати під аналітику даних.

Було наглядно продемонстровано роботу платформи за допомогою прикладу, в якому моделюється робота системи.

4 ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ ДЛЯ ВИЯВЛЕННЯ АВАРІЙНИХ СИТУАЦІЙ НА ПІДПРИЄМСТВІ ШЛЯХОМ ВПРОВАДЖЕННЯ ПЛАТФОРМИ ІНТЕРНЕТУ РЕЧЕЙ

4.1 Постановка задачі проектування

Спроектувати програмний продукт для виявлення аварійних ситуацій на підприємтсі. Він призначений для використання в середовищах що підтримують технології Java Virtual Machine

4.2 Обґрунтування функцій та параметрів програмного продукту

Виходячи з конкретних цілей, які реалізуються :

F1 – мова програмування:

а) Java,

б) Python.

F2 – протокол:

а) MQTT,

б) REST.

F3 – обробка даних:

а) Apache Flink,

б) Apache Spark.

Виходячи з представлених варіантів будуємо морфологічну карту (рис.4.1).

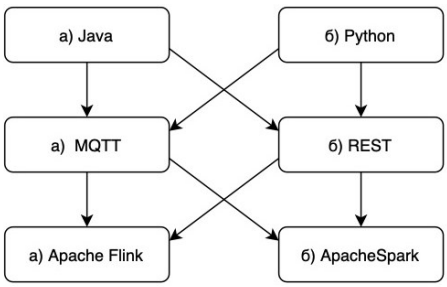


Рисунок 4.1 Морфологічна карта

Спираючись на карту була побудована позитивно-негативна матриця таблиця 4.1

Таблиця 4.1 Позитивно-негативна матриця

Основна функція	Варіант реалізації	Переваги	Недоліки
F1	А	Продуктивність, статична типізація, JIT (Just-in-Time Compiler)	Boilerplate code
	Б	Легкість використання	Динамічна типізація
F2	А	Негайне оновлення даних, розміщених на каналі	Завищені вимоги до продуктивності
	Б	Спільність інтерфейсів, незалежне впровадження компонентів	Не існує чіткого визначення
F3	А	Час затримки	Підтримка спільноти
	Б	Надійність, масштабованість і відмовостійкість	Час затримки, режим роботи з даними

Для характеристики прототипу програмного додатку використовуємо параметри X1 – X4. На основі даних про основні вимоги, які повинен задовільняти програмній продукт, визначаємо мінімальні, середні отримуванні та максимально допустимі значення таблиця 4.2.

Таблиця 4.2 Система параметрів додатку

Найменування параметру	Позначення параметру	Значення параметру		
		Мінімальне	Середнє	Максимальне
Час розробки, людина*год	X1	352	528	704
Час обробки даних, мс	X2	100	500	1000
Об’єм коду, ряд	X3	2000	5000	10000
Час обробки результату, мс	X4	10	60	110

За даними табиці 4.2 побудовано графічні характеристики параметрів – рис.4.2 – рис.4.5.

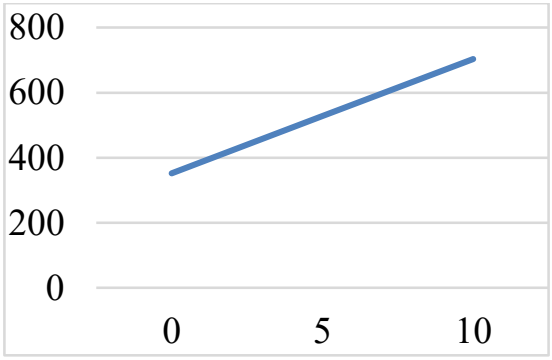


Рисунок 4.2 Значення параметру X1

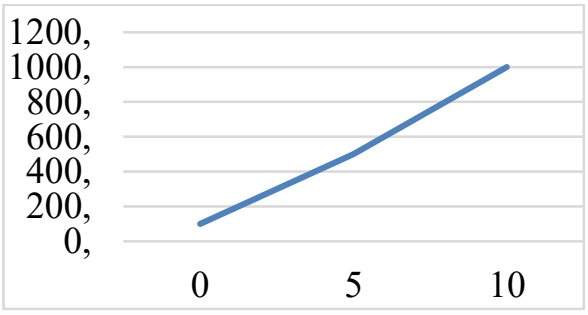


Рисунок 4.3 Значення параметру X2

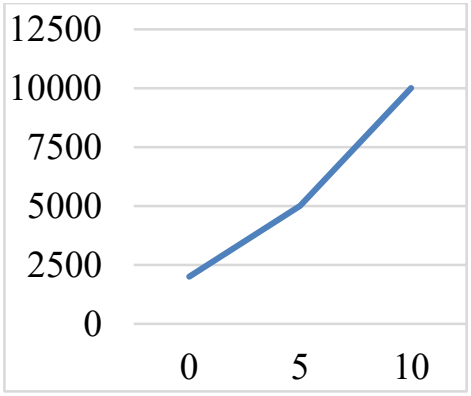


Рисунок 4.4 Значення параметру X3

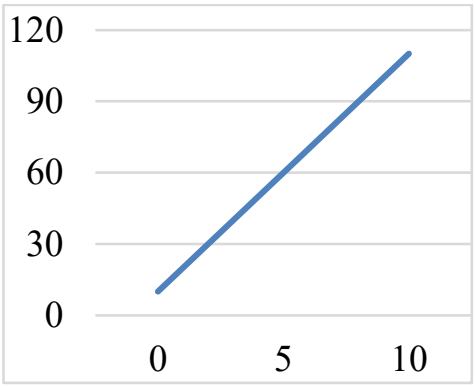


Рисунок 4.5 Значення параметру X4

Вагомість параметрів оцінюється за допомогою методів попарного зрівняння. Ранги варіюються від 1 до 4. Оцінку проводити експертна комісія Із 7 людей. Результати наведені в таблиця 4.3-4.4

Визначемо коефіцієнт конкординації:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 177}{7^2(4^3 - 4)} = 0.72 > W_k = 0.67 \quad (4.1)$$

Так як коефіцієнт узгодженості більше нормативного, результати вважають достовірними.

Таблиця 4.3 Результат оцінки параметрів

Параметр	Ранг параметру по оцінці експерта							Сума рангів, R_i	Відхилення Δ_i	Квадрат відхилення, $(\Delta_i)^2$
	1	2	3	4	5	6	7			
X1	4	3	3	3	4	2	3	22	4,5	20.25
X2	3	4	4	4	3	4	4	26	8,5	72.25
X3	1	1	1	2	1	3	2	11	-6,5	42.25
X4	2	2	2	1	2	1	1	11	-6,5	42.25
Разом	10	10	10	10	10	10	10	70	0	177

Таблиця 4.4 Попарне зрівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 та X2	>	<	<	<	>	<	<	<	0.5
X1 та X3	>	>	>	>	>	<	>	>	1.5
X1 та X4	>	>	>	>	>	>	>	>	1.5
X2 та X3	>	>	>	>	>	>	>	>	1.5
X2 та X4	>	>	>	>	>	>	>	>	0.5
X3 та X4	<	<	<	>	<	>	>	<	0.5

Розрахунок вагомості параметрів наведено в таблиці 4.5

Таблиця 4.5 Розрахунок вагомості параметрів

Параметри xi	Параметри xj				Перший крок		Другий крок		Третій крок	
	X1	X2	X3	X4	b_i	$К_{вi}$	b_i	$К_{вi}$	b_i	$К_{вi}$
X1	1	0,5	1,5	1,5	4,5	0,28	16,25	0,28	59,13	0,27
X2	1,5	1	1,5	1,5	5,5	0,34	21,25	0,36	77,88	0,36
X3	0,5	0,5	1	0,5	2,5	0,16	9,25	0,16	34,13	0,16
X4	0,5	0,5	1,5	1	3,5	0,22	12,25	0,20	44,88	0,20
Загалом:					16	1	59	1	216	1,00

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основна функція	Варіант реалізації	Абсолютне значення параметру	Бальна оцінка параметру	Коефіцієнт вагомості параметру	Коефіцієнт якості
F1(X1)	а)	400	2	0,27	0,54
F1(X3)	а)	7000	7	0,16	1,12
F2(X4)	а)	12	0,5	0,20	0,1
F3(X2)	а)	150	1	0,36	0,36
	б)	200	2,5		0,9

Обрахуємо коефіцієнти якості кожного з варіантів розробки за даними за таблиці 4.6:

$$K_{я1} = 0{,}54 + 1{,}12 + 0{,}1 + 0{,}36 = 2.12 \quad (4.2)$$

$$K_{я2} = 0{,}54 + 1{,}12 + 0{,}1 + 0{,}9 = 2.66 \quad (4.3)$$

Оскільки варіант 2 має найбільший коефіцієнт якості, він є найкращим.

4.3 Економічний аналіз варіантів розробки

Для оцінки вартості розробки спочатку проведемо розрахунок трудомісткості. Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної реалізації платформи.

Також кожний з варіантів має два додаткових завдання, які є реалізаціями розгалужених варіантів розробки незалежного модуля. Далі наведено варіанти додаткових завдань(два завдання, які мають номери 3 в реалізаціях та два завдання, які мають номери 4 в реалізаціях)

3. Оптимізація використання диску при обробці даних.
4. Реалізація алгоритму обробки даних.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 2.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Спираючись на норми розрахункового часу визначимо трудомісткість. Вона складає для першого завдання $T_р$=90 людино-днів. Поправочний коефіцієнт складає $К_п$=1,7 (нормативно-довідкова інформація). Оскільки під час виконання даного завдання використовуються новостворенні модулі, врахуємо це за допомогою коефіцієнта $К_{ст}$ = 1. Коефіцієнти $К_м$ і $К_{ст.п}$, які враховують відповідно

програмування на мові низького рівня та розробку стандартного програмного забезпечення, для всіх семи завдань дорівнюють 1.

Повна трудомісткість першого завдання(складність – 1, новизна – А):

$$T_1 = 90 * 1,7 * 0,8 = 122,4 \quad (4.4)$$

Аналогічно для завдання 2 (складність – 2, новизна – Б):

де $T_р$= 27;

$К_п$ = 0,9;

$К_{ст}$ = 0,8;

Аналогічно для завдання 3(складність – 3, новизна – Г):

$$T_3 = 8 * 0,3 * 0,8 = 1,92 \quad (4.6)$$

де $T_р$= 8;

$К_п$ = 0,6;

$К_{ст}$ = 0,8;

Аналогічно для завдання 4(складність – 2, новизна – В)

$$T_4 = 19 * 0,6 * 0,8 = 9,12 \quad (4.7)$$

де $T_р$= 19;

$К_п$ = 0,36;

$К_{ст}$ = 0,6;

Визначимо повну трудомісткість варіантів(людино-днів):

$$T_1 = 122,4 + 19,44 + 1,92 = 145,76 \quad (4.8)$$

$$T_2 = 122,4 + 19,44 + 9,12 = 152,96 \quad (4.9)$$

Найбільш трудомісткими завданнями є 2 .

Далі вважається, що робочий день складає 8 годин, в тижні п’ять робочих днів. В розробці бере участь один програміст з окладом 11000 грн та тестувальник з окладом 12000 грн . Визначимо середню заробітну плату за годину:

$$C_ч = \frac{11000 + 12000}{2 * 22 * 8} = 65,34 \quad (4.10)$$

Тоді заробітна плата для кожного з варіантів реалізації(грн):

$$С_{ЗП} = 65,34 * 8 * 145,76 = 76191,70 \quad (4.11)$$

$$С_{ЗП} = 65,34 * 8 * 152,96 = 79955,25 \quad (4.12)$$

Відрахування на єдиний соціальний внесок становить 22%(грн):

$$С_{ВІД} = 76191,70 * 0,22 = 16762,17 \quad (4.13)$$

$$С_{ВІД} = 79955,25 * 0,22 = 17590,15 \quad (4.14)$$

Далі розрахуємо витрати на оплату однієї машино-години. Враховуючи, що вона обслуговує одного спеціаліста з окладом 11000 грн та одного з окладом 12000 грн з коефіцієнтом зайнятості 0,5 , то для двох машин отримаємо

$$С_{г} = 12 * 11000 * 0,5 + 12 * 12000 * 0,5 = 138000 \text{ грн} \quad (4.15)$$

Враховуючи додаткову заробітну плату

$$С_{ЗП} = 138000 * (1 + 0,4) = 193200 \quad (4.16)$$

Відрахування на соціальне страхування 22%

$$С_{ВІД} = 193200 * 0,22 = 42504 \quad (4.17)$$

Розрахуємо амортизаційні підрахунки (амортизація 25%, вартість ЕОМ – 20000 грн.)

$$С_{А} = К_{ТМ} * К_{А} * Ц_{ПР} = 1,15 * 0,25 * 20000 = 5750 \text{ грн} \quad (4.18)$$

Розрахуємо витрати на ремонт та профілактику:

$$С_{Р} = К_{ТМ} * Ц_{ПР} * К_{Р} = 1,15 * 20000 * 0,05 = 1150\text{грн} \quad (4.19)$$

Розрахуємо ефективний годинний фонд часу ПК за рік

$$Т_{ЕФ} = (365 - 142 - 16) * 8 * 0,8 = 1324,8\text{год} \quad (4.20)$$

Розрахуємо витрати на електроенергію

$$С_{ЕЛ} = 1324,8 * 0,5 * 0,5 * 1,75 = 579,6\text{грн} \quad (4.21)$$

Накладні витрати рівні:

$$С_{Н} = 20000 * 0,67 = 13400 \text{ грн.} \quad (4.22)$$

Отже експлуатаційні витрати(грн):

$$С_{ЕКС} = 193200 + 42504 + 5750 + 1150 + 579,6 + 13400 = 255983,6 \quad (4.23)$$

Тоді собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М-Г} = \frac{255983,6}{1324,8} = 193,22 \text{грн/год} \quad (4.24)$$

Враховуючи, що всі роботи ведуться на ЕОМ, витрати на оплату машинного часу:

$$С_{м} = 193,22 * 8 * 145,76 = 225309,98 \quad (4.25)$$

$$С_{м} = 193,22 * 8 * 152,96 = 236439,45 \quad (4.26)$$

Накладні витрати відповідно

$$С_{Н} = 76191,70 * 0,67 = 51048 \quad (4.27)$$

$$С_{Н} = 79955,25 * 0,67 = 53570 \quad (4.28)$$

Розрахуємо повну вартість розробки за варіантами:

$$С_{ПП} = 76191,70 + 16762,17 + 225309.98 + 51048 = 369311,85 \quad (4.29)$$

$$С_{ПП} = 79955,25 + 17590,15 + 236439,45 + 53570 = 387554,85 \quad (4.30)$$

4.4 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня

$$К_{ТЕР1} = \frac{2,12}{369311,85} = 0,57 * 10^{-5} \quad (4.31)$$

$$К_{ТЕР2} = \frac{2,66}{387554,85} = 0,68 * 10^{-5} \quad (4.32)$$

4.5 Висновки до розділу 4

Проведено економічні розрахунки, які спрямовані на визначення економічної ефективності та вартості проведення дослідження.

Отже враховуючи всі дослідження, що описані вище, можна сказати, що 2 варіант реалізації є найбільш оптимальним зі сторони якісно-економічної оцінки. Його коефіцієнт техніко-економічного рівня складає $0,68*10^{-5}$.

Цей варіант реалізації програмного продукту выдповідає таким параметрам: мова програмування – Java, протокол передачі даних – mqtt, фреймворк обробки даних – Apache Spark.

ВИСНОВКИ

В цій дипломній роботі було досліджено та розроблено програмний продукт, який дозволяє ідентифікувати аварійну ситуацію на підприємстві.

Було проаналізовано та сформовано вимоги, що висуваються до систем інтернету речей. Враховано основні критерії, якими повинна володіти сучасна платформа.

На основі вимог, розроблено та реалізовано програмний продукт для обробки великих за обсягом задач.

Вивчені сучасні засоби передачі інформації в інтернеті речей. Проаналізовані протоколи та методи, що використовуються для передачі інформації в Інтернеті речей та дозволило ідентифікувати позитивні та негативні якості кожного з протоколів, та обрати найкращі для створення мережі.

Платформа, що була розроблена повністю задовольняє вимогам та містить в собі реалізацію наступних систем: приймання, зберігання, обробку операційних даних; автентифікацію, моніторинг. Дані системи дозволяють використовувати платформу таким групам користувачів як: пристрої, адміністратори і адміністратори безпеки.

У роботі запропонований та реалізований сучасний підхід у вигляді мікросервісної архітектури, яка дозволяє працювати компонентам незалежно, швидко та має високі продуктивність та рентабельність.

Реалізується зворотній зв’язок від платформи до користувача.

Платформа розподілена та масштабована при цьому застосовується механізм гарантованної доставки інформації, що знижу ймовірність втрати даних та забезпечує майже повну доступність.

Рішення засноване на використанні сторонніх продуктах із відкритим вихідним кодом зумовлене тим, що система набуває більшої гнучкості, підвищеною ефективністю. Також, завдяки сторонім рішенням, поліпшена сигментація.

Однією з говних відміностей запропонованої системи є горизонтальна масштабованість, яка дозволяє розширити систему шляхом додавання компонентів – сервісів. Також запропонована платформа легко адаптується до системи, що може обробляти та аналізувати вхідні великі дані.

Робота платформ була продемонтрована на прикладі сервісів, які виконують основну фнкціональність: датчик, що симулює відправку даних, їх потрапляння на платформу та оброка, отримання повідомлення на пошту в якості результату.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Доповідь компанії Cisco. URL: http://www.cisco.com (дата звернення: 23.05.2020).

2. N. Marz, J. Warren. Big Data: Principles and best practices of scalable realtime data systems. Manning Publications, 2015. С. 100-170.

3. A. Sathi. Big Data Analytics: Disruptive Technologies for Changing the Game. Mc Press, 2012. С. 60-93.

4. Vijay K. Garg. Elements of Distributed Computing. WileyIEEE Press, 2002. С. 320-400.

5. G. Tel. Introduction to Distributed Algorithms. Cambridge University Press, 2000. С. 400-490.

6. M. Tamer Özsu, Patrick Valduriez, M. Tamer Özsu. Principles of Distributed Database Systems. Springer, 2011. С. 400-800.

7. M. Fowler, K. Beck, J. Brant. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, 1999. С. 150-273.

8. Martin R. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008. С. 192-308.

9. Gamma E., Helm R, Johnson R. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994. С. 174-200.

10. Fowler M.. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. С. 247- 476.

11. Bass L., Clements L, Kazman R. Software Architecture in Practice Addison-Wesley Professional, 2012. С. 290-500.

12. Hohmann R. Beyond Software Architecture: Creating and Sustaining Winning Solutions. Addison -Wesley Professional, 2003. С. 150-200.

13. Інформація про платформу AWS IoT. URL: https://aws.amazon.com (дата звернення: 24.05.2020).

14. Інформація про платформу IBM IoT Platform. URL: http://www.ibm.com (дата звернення: 25.05.2020).

15. Інформація про платформу Oracle IoT. URL: https://cloud.oracle.com (дата звернення: 25.05.2020).

16. Специфікація HTTP/2 (Internet Engineering Task Force. Request for Comments 7540). URL: https://tools.ietf.org (дата звернення: 26.05.2020).

17. Специфікація WebSocket (Internet Engineering Task Force. Request for Comments 6455). URL: https://tools.ietf.org (дата звернення: 27.05.2020).

18. Perera C., Liu C., Jayawardena S. The Emerging Internet of Things Marketplace From an Industrial. 2015. C. 5-10.

19. Survey A. IEEE Transactions on Emerging Topics in Computing. C. 3.

20. Office of Senator Mark Warner Bipartisan Legislation to Improve Cybersecurity of Internet-of-Things Devices Introduced in Senate & House. (Press release). March 11, 2019. C. 10-12.

21. Tschider C. Regulating the IoT: Discrimination, Privacy, and Cybersecurity in the Artificial Intelligence Age. 2018

22. Interagency International Cybersecurity Standardization Working Group, Interagency Report on Status of International Cybersecurity Standardization for the Internet of Things (IoT). NIST. November 2018

23. Getting started with mqtt guide URL: https://docs.wixstatic.com (дата звернення: 19.05.2020).

24. Kafka basic training. URL: https://ondemand.cloudera.com (дата звернення: 19.05.2020).

25. Kafka developer training URL: https://ondemand.cloudera.com (дата звернення: 19.05.2020).

26. Doctor Kafka URL: http://github.com (дата звернення: 19.05.2020).

27. Kafka manager URL: http://github.com (дата звернення: 19.05.2020).

28. Newman S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015. C. 92-150.

29. Офіційна документація Apache Kafka. URL: http://kafka.apache.org (дата звернення: 29.05.2020).

30. Офіційна документація Apache Spark. URL: http://spark.apache.org. (дата звернення: 26.05.2020).

31. Інформація про Sort Benchmark і його результати. URL: http://sortbenchmark.org. (дата звернення: 31.05.2020).

32. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing / [Matei Zaharia, Mosharaf Chowdhury, Tathagata Das etal.]. URL: https://www.cs.berkeley.edu (дата звернення: 29.05.2020).

33. Офіційна документація Apache Cassandra. URL: https://wiki.apache.org. (дата зверення: 25.05.2020).

34. Dynamo: Amazon’s Highly Available Key-value Store / [Giuseppe DeCandia, Deniz Hastorun, Madan Jampani et al.]. URL: http://www.allthingsdistributed.com (дата звернення: 02.06.2020).

35. Provos N., Mazières D. A Future-Adaptable Password Scheme Proceedings of 1999 USENIX Annual Technical Conference 1999. С. 81-92.

36. Офіційна документація. URL: http://dev.mysql.com (дата звернення: 28.05.2020).

37. Специфікація JSON Web Token (Internet Engineering Task Force. Request for Comments 7519). URL: https://tools.ietf.org (дата звернення: 26.05.2020).

38. Офіційна документація HAProxy. URL: http://www.haproxy.org. (дата зврнення: 25.05.2020).

39. Офіційна документація Amazon Web Services. URL: https://aws.amazon.com. (дата звернення: 25.05.2020).

40. Офіційна документація Docker. URL: https://www.docker.com. (дата звернення: 25.05.2020).

41. Офіційна документація Consul. URL: https://www.consul.io. (дата звернення: 25.05.2020)

ДОДАТОК А

Оброблені дані, які потрапляють в Kafka events

{"endpointId":"40fbcf5b-02d9-4614-82d1-b68b18cc43e7","dataSample":{"data":{"counter":"7","humidity":"75","sinCurve":"38.1084463945195","speed":"90","temperature":"25","weight":"300"},"timestamp":1590767049991,"~ep-metadata":{"deviceModel":"MyDevice A300","equipmentModel":"V8","name":"TheMostBeautifulDevice","role":"Flagman","softwareVersion":"v0.0.1","timeZone":"+02:00","type":"The best type"}},"serverTimestamp":1590767051037}

{"endpointId":"40fbcf5b-02d9-4614-82d1-b68b18cc43e7","dataSample":{"data":{"counter":"8","humidity":"75","sinCurve":"49.252136217971895","speed":"90","temperature":"25","weight":"300"},"timestamp":1590767050991,"~ep-metadata":{"deviceModel":"MyDevice A300","equipmentModel":"V8","name":"TheMostBeautifulDevice","role":"Flagman","softwareVersion":"v0.0.1","timeZone":"+02:00","type":"The best type"}},"serverTimestamp":1590767052038}

{"endpointId":"40fbcf5b-02d9-4614-82d1-b68b18cc43e7","dataSample":{"data":{"counter":"9","humidity":"75","sinCurve":"46.83364672796234","speed":"90","temperature":"25","weight":"300"},"timestamp":1590767051991,"~ep-metadata":{"deviceModel":"MyDevice A300","equipmentModel":"V8","name":"TheMostBeautifulDevice","role":"Flagman","softwareVersion":"v0.0.1","timeZone":"+02:00","type":"The best type"}},"serverTimestamp":1590767053037}

{"endpointId":"40fbcf5b-02d9-4614-82d1-b68b18cc43e7","dataSample":{"data":{"counter":"10","humidity":"75","sinCurve":"31.51893864518064","speed":"90","temperature":"25","weight":"300"},"timestamp":1590767052991,"~ep-metadata":{"deviceModel":"MyDevice A300","equipmentModel":"V8","name":"TheMostBeautifulDevice","role":"Flagman","

softwareVersion":"v0.0.1","timeZone":"+02:00","type":"The best type"}},
"serverTimestamp":1590767054036}

{"endpointId":"40fbcf5b-02d9-4614-82d1-
b68b18cc43e7","dataSample":{"data":{"counter":"11","humidity":"75","sinCurve":"7.5
25104528471391","speed":"90","temperature":"25","weight":"300"},"timestamp":1590
767053991,"~ep-metadata":{"deviceModel":"MyDevice
A300","equipmentModel":"V8","name":"TheMostBeautifulDevice","role":"Flagman","
softwareVersion":"v0.0.1","timeZone":"+02:00","type":"The best
type"}},"serverTimestamp":1590767055036}

{"endpointId":"40fbcf5b-02d9-4614-82d1-
b68b18cc43e7","dataSample":{"data":{"counter":"12","humidity":"75","sinCurve":"-
18.54085934350385","speed":"90","temperature":"25","weight":"300"},"timestamp":15
90767054991,"~ep-metadata":{"deviceModel":"MyDevice
A300","equipmentModel":"V8","name":"TheMostBeautifulDevice","role":"Flagman","
softwareVersion":"v0.0.1","timeZone":"+02:00","type":"The best type"}},
"serverTimestamp": 1590767056036}

ДОДАТОК Б

Kafka в topic rule

{"name":"TheMostBeautifulDevice","type":"SQL","timeZone":"+02:00","equipmentModel":"V8","deviceModel":"MyDevice A300","serialNumber":"40fbcf5b-02d9-4614-82d1-b68b18cc43e7","timestamp":1590767055036,"data":{"data":{"counter":"11","humidity":"75","sinCurve":"7.525104528471391","speed":"90","temperature":"25","weight":"300"},"softwareVersion":"v0.0.1","partitionOffset":"1-4565"}

{"name":"TheMostBeautifulDevice","type":"SQL","timeZone":"+02:00","equipmentModel":"V8","deviceModel":"MyDevice A300","serialNumber":"40fbcf5b-02d9-4614-82d1-b68b18cc43e7","timestamp":1590767056036,"data":{"data":{"counter":"12","humidity":"75","sinCurve":"-18.54085934350385","speed":"90","temperature":"25","weight":"300"},"softwareVersion":"v0.0.1","partitionOffset":"1-4566"}

Додаток В

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Дипломна робота на тему:
Виявлення аварійних ситуацій на підприємстві шляхом впровадження технологій інтернету речей

Студентка: Хархута М.С.
Науковий керівник: Кухарєв С.О.

1

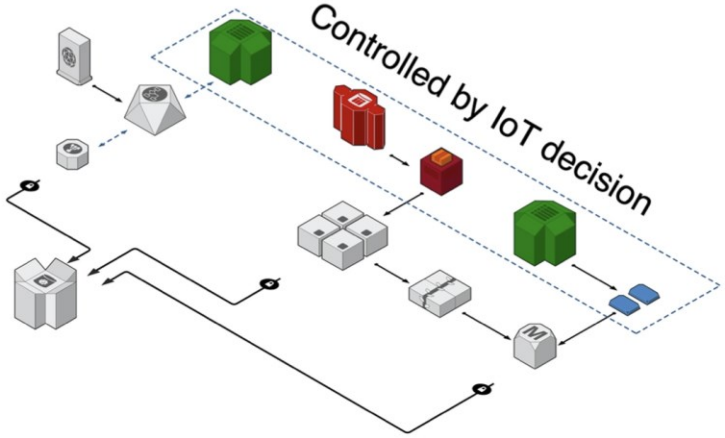
Постановка задачі

- Мета роботи: Розробка програмного продукту який дозволяє ідентифікувати аварійну ситуацію
- Об’єкт дослідження: Обробка даних, які генерує розумний датчик та керування елементами інтернету речей
- Предмет дослідження: Засоби та протоколи даних в мережі інтернету речей і системи управління її елементами

2

Актуальність

IoT створює безперебійну комунікацію між пристроями та працівниками.



3

Аварійна ситуація

Аварійна ситуація — стан потенційно небезпечного об'єкта, що характеризується порушенням меж та (чи) умов безпечної експлуатації

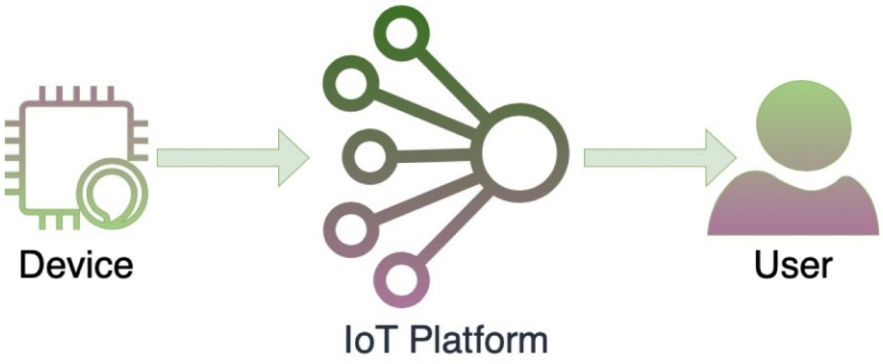
4

Існуючі рішення

- логіко - імовірнісні
- теоретично - імовірнісні
- статистичні методи
- метод "древо подій"
- метод "древо відмов"
- IoT рішення

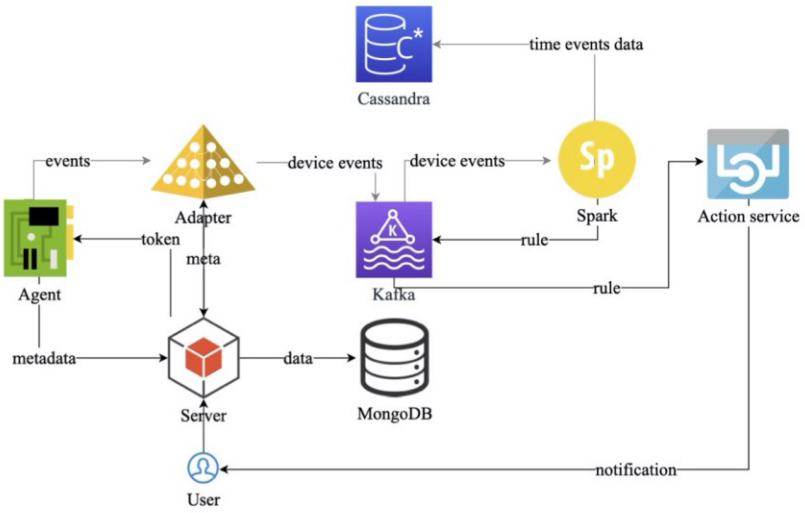
5

IoT - рішення



6

Використані технології:



7

Бази даних

8

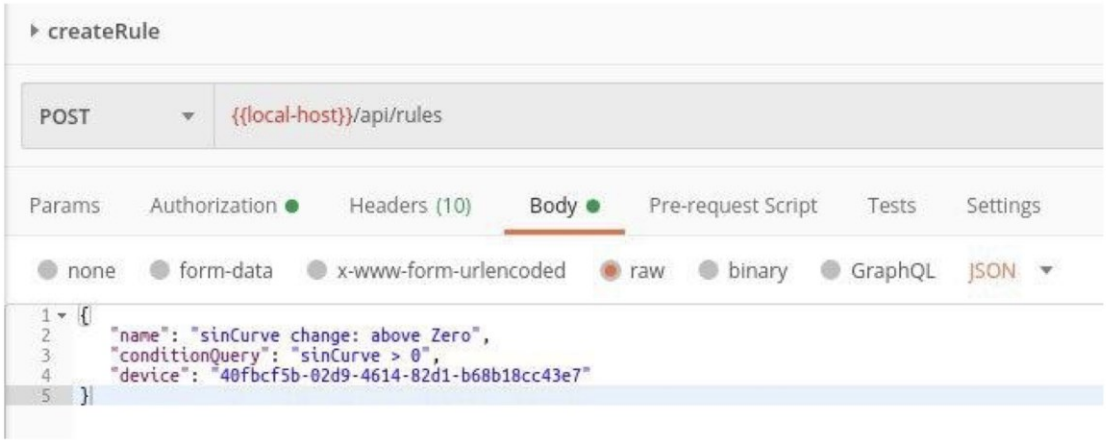
Брокер повідомлень

9

Обробка даних

10

Створення умови



11

Відправлення даних


```
Successfully delivered bucket 5
Sending metrics by [smyYJKF1b8] to topic {v1/srv/events}
Sent metrics: [{"timestamp":1590768886764,"data":{"sinCurve":"49.384654417694705","temperature":"25","weight":"300","humidity":"75","counter":"6","speed":"90"}}]
Successfully delivered bucket 6
Sending metrics by [smyYJKF1b8] to topic {v1/srv/events}
Sent metrics: [{"timestamp":1590768887764,"data":{"sinCurve":"38.62542661635164","temperature":"25","weight":"300","humidity":"75","counter":"7","speed":"90"}}]
Successfully delivered bucket 7
Sending metrics by [smyYJKF1b8] to topic {v1/srv/events}
Sent metrics: [{"timestamp":1590768888764,"data":{"sinCurve":"17.23821422519136","temperature":"25","weight":"300","humidity":"75","counter":"8","speed":"90"}}]
Successfully delivered bucket 8
Sending metrics by [smyYJKF1b8] to topic {v1/srv/events}
Sent metrics: [{"timestamp":1590768889764,"data":{"sinCurve":"-8.9895488164671864","temperature":"25","weight":"300","humidity":"75","counter":"9","speed":"90"}}]
```


12

Отримання повідомлення

Висновки

- Запропоновано та реалізовано таку мікросервісну платформу, яка дозволяє ідентифікувати аварійну ситуацію на підприємстві
- Розроблене рішення є добре розширюваним у тому сенсі, що його можливо легко адаптувати під аналітику даних
- Було наглядно продемонстровано роботу платформи за допомогою прикладу, в якому моделюється робота системи

Дякую за увагу!