Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційних радіотехнологій та технічного захисту інформацій

Кафедра Радіотехнологій інформаційно-комунікаційних систем

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

ГЮІК. 467750.004 ПЗ
(позначення документа)

Розробка платформи дистанційного контролю параметрів віддалених об'єктів на прикладі метеостанції
(тема)

Виконав:
студент II курсу, групи РПСКм -18-1
Болдиш М. І.
(прізвище, ініціали)

Спеціальність
172 Телекомунікації та радіотехніка
(код і повна назва спеціальності)

Тип програми
освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Радіоелектронні пристрої, системи та комплекси
(повна назва освітньої програми)

Керівник доцент Бітченко О.М.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри РТІКС _____ Цопа О.І.
(підпис) (прізвище, ініціали)

2019 р.

Не містить відомостей заборонених для відкритого публікування.

Студент М.І. Болдиш

Керівник О.М. Бітченко

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет Інформаційних радіотехнологій та технічного захисту інформацій
Кафедра Радіотехнологій інформаційно-комунікаційних систем
Рівень вищої освіти другий (магістерський)
Спеціальність 172 Телекомунікації та радіотехніка
Тип програми Освітньо-професійна
Освітня програма Радіоелектронні пристрої, системи та комплекси

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2019 р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові **БОЛДИШУ Максиму Ігоровичу**
(прізвище, ім'я, по батькові)

1. Тема роботи **РОЗРОБКА ПЛАТФОРМИ ДИСТАНЦІЙНОГО КОНТРОЛЮ ПАРАМЕТРІВ ВІДДАЛЕНИХ ОБ'ЄКТІВ НА ПРИКЛАДІ МЕТЕОСТАНЦІЇ**

затверджена наказом по університету від **21 листопада 2019 р. № 1729Ст**

2. Термін подання студентом проекту (роботи) **12 грудня 2019 р.**

3. Вихідні дані до проекту (роботи)
3.1 Данні з метеостанції передаються бездротовим шляхом .
3.2 Час відправки данних з метеостанції не більше 30 секунд .

4. Перелік питань, що потрібно опрацювати в роботі
Реферат. Перелік умовних позначень, символів, одиниць, скорочень і термінів. Вступ. 4.1 Огляд та аналіз аналогічних рішень. 4.2 Великі дані. 4.3 Розробка архітектури платформи. 4.4 Побудова платформи. 4.5 Розробка метеостанції. 4.6 Розробка веб-серверу. Перелік посилань. Додатки. .

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)

Комп'ютерна презентація

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по-батькові	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	доц. Бітченко Олександр Миколайович		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Вступ	5.09-10.09	Виконано
2	Огляд та аналіз аналогічних рішень	10.09-10.20	Виконано
3	Великі дані	10.20-10.25	Виконано
4	Розробка архітектури платформи	10.26-11.01	Виконано
5	Побудова платформи	11.02-11.15	Виконано
6	Розробка метеостанції	11.16-11.24	Виконано
7	Розробка веб-серверу	11.25-12.01	Виконано
8	Висновки	12.02-12.03	Виконано
9	Оформлення пояснювальної записки	12.04-12.06	Виконано
10	Оформлення презентації	12.07-12.08	Виконано
11	Подання роботи на кафедру	9.12.2019	Виконано

Дата видачі завдання **4 вересня 2019 р**

Студент _____
(підпис)

Керівник роботи_____ доц. Бітченко О.М.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Магістерська атестаційна робота складається з пояснювальної записки, котра містить: 120 сторінок тексту, 40 рисунків, 61 джерело та 5 додатки.

МЕТЕОСТАНЦІЯ.МЕТРОЛОГІЯ.
ДАНІ.СЕРВЕР.ARDUINO.GCP.BIGQUERY.PUB/SUB

Об'єкт роботи – платформа дистанційного контролю параметрів віддалених об’єктів та метеостанція.

Мета роботи – схемотехнічна та конструкторська розробка платформи дистанційного контролю параметрів віддалених об’єктів та метеостанції.

Для досягнення поставленої мети проаналізовані методи роботи з великими даними, розроблена структурна схема платформи, обґрунтовано вибір компонентів для реалізації платформи, розроблена метеостанція.

THE ABSTRACT

The master’s degree work consists of an explanatory note, which contains: 120 pages of text, 40 figures, 61 sources and 5 appendices.

WEATHERSTATION.
METROLOGY.DATA.SERVER.ARDUINO.GCP.BIGQUERY.PUB/SUB

The object of Work - Remote object parameters control platform and weather station.

Goal of the work - schematic and design development of a platform for remote control of parameters of remote objects and a weather station.

To achieve this goal, the methods of work with big data are analyzed, the structural scheme of the platform is developed, the choice of components for the platform implementation is grounded, the weather station is developed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ID – ідентифікатор;

API – прикладний програмний інтерфейс;

GSM – глобальний стандарт цифрового мобільного сотового зв’язку;

GPRS – пакетний радіозв’язок спільного використання;

UART – універсальный асинхронний приймач;

USB – універсальная послідовна шина;

TCP/IP – мережева модель передачі цифрових даних;

REST – передача репрезентативного стану;

АЦП – аналогово-цифровий перетворювач.

ВСТУП

У сучасному світі все більшого розвитку набуває інтернет речей. Інтернет речей – це концепція мережі, що складається із взаємозв'язаних фізичних пристроїв, які мають вбудовані сенсори (датчики), а також програмне забезпечення, що дозволяє здійснювати передачу і обмін даними між фізичним світом і комп'ютерними системами, за допомогою використання стандартних протоколів зв'язку.

Окрім датчиків, мережа може мати виконавчі пристрої, вбудовані у фізичні об'єкти і пов'язані між собою через дротові чи бездротові мережі. Ці взаємопов'язані пристрої мають можливість зчитування та приведення в дію, функцію програмування та ідентифікації, а також дозволяють виключити необхідність участі людини, за рахунок використання інтелектуальних інтерфейсів [1].

Все більш звичним явищем стає використання “розумних” приладів у побуті. Так, зараз використовують “розумні” сенсори, лампочки, двері, тощо. Це явище має назву розумний дім. Розумний дім – це будинок, дача або приміщення комерційного призначення (бутік, офіс, будь-яка установа), які мають якісні системи забезпечення та операційний multi-room. За допомогою останнього, функціонально пов'язуються між собою усі електроприлади будівлі, якими можна керувати централізовано — з пульта-дисплею.

Прилади можуть бути під'єднані до комп'ютерної мережі, що дозволяє керувати ними за допомогою ПК та надає віддалений доступ до них через Інтернет. Завдяки інтеграції інформаційних технологій у домашні умови, усі системи та прилади узгоджують виконання функцій між собою, порівнюючи задані програми та зовнішні показники (обстановки) [2].

Вже зараз можна, наприклад, придбати розумну лампу та завантажити додаток на свій смартфон після чого можна за допомогою свого смартфону її включати або виключати. Окрім розумного дому інтернет речей також використовується в промисловій діяльності. Так наразі розумні пристрої

використовується майже у всіх напрямках промисловості будь то медицина, металургія чи агрономія.

Основними функціями які виконують такі девайси є виконання команд та збирання даних. Через те, що данні з великої кількості девайсів поступають дуже часто з’явилася потреба в правильній обробці та збереженні цих даних. Це призвело до створення такого поняття як великі данні. Великі данні – це набори інформації (як структурованої, так і неструктурованої) настільки великих розмірів, що традиційні способи та підходи (здебільшого засновані на рішеннях класу бізнесової аналітики та системах управління базами даних) не можуть бути застосовані до них. Альтернативне визначення називає великими даними феноменальне прискорення нагромадження даних та їх ускладнення [3].

Налаштування та робота з промисловими девайсами потребує великих зусиль. Ще однією проблемою є те, що різні девайси можуть працювати в різних мобільних чи браузерних додатках, що значно ускладнює процес їх використання.

Зараз почали розвиватися платформи які надають можливість працювати з різними девайсами в одному місці. Однак, як правило кількість датчиків які надають подібні платформи є невеликим. Іншим прикладом, є платформи які надають можливість будувати свої власні рішення на базі таких платформ за допомогою бібліотек. Це дозволяє використовувати практично будь які девайси та сенсори. Також такі платформи дозволяють інтегруватися з хмарними рішеннями для подальшої обробки та збереження інформації якщо рішення які пропонує та чи інша платформа є недостатнім.

Як приклад приладу в даній роботі взято метеостанцію. У сучасно світі найпопулярнішим способом отримання інформації про погоду на добу і тижні є централізовані метеорологічні станції. За допомогою них можна отримати детальнішу інформацію про погодні умови в найближчому майбутньому по всій планеті. Інформацію з цих центрів можна отримати через сучасні засоби комунікації, такі як телевізор, радіо та інтернет. Але є у таких прогнозів і свої недоліки. Так як інформація про погодні умови збирається для великих

регіонів, то і інформація про поточну погоду буде загальною для конкретного міста, не враховуючи кліматичні стану в конкретних районах.

Хоч централізовані метеорологічні станцій і є найпопулярнішим засобом отримання інформації про погоду в наші дні, все більшою популярністю починають користуватися домашні метеостанції. Такі пристрої дозволяють вимірювати погодні дані, обробляти їх і робити прогноз погодних умов на дні і тижні. Одним з найважливіших переваг такого типу пристроїв є те, що вони показують інформацію про погоду в конкретному місці, де вони встановлені, а не загальну як це роблять централізовані станції. Це дозволяє оцінювати погодні умови на конкретній ділянці, а не покладатися на загальну інформацію для регіону.

Будь-яка домашня метеостанція містить в своєму складі датчик температури та повітря. Також обов'язковим для таких метеостанцій датчику тиску. За допомогою датчика тиску можна визначати тенденції зміни погоди. Зараз, більшість домашніх метеостанцій роблять модульними, що дозволяє додавати до базових датчикам, датчики, які вимірюють додаткові метеорологічні величини.

Інший вид метеостанцій - метеостанції для віддаленого спостереження за погодними умовами. Такі метеостанції використовуються в аграрному, лісовому та багатьох інших секторах. Вони дозволяють вимірювати температуру, вологість і дозволяють будувати прогноз на найближче майбутнє. Як правило, такі метеостанції підключаються до інтернету через свій центральний модуль, який в свою чергу приєднується до мережі інтернет через маршрутизатор. До центрального модулю підключаються додаткові модулі для вимірювання погодних умов. Додаткові модулі підключаються через Ethernet або по радіосигналу. Всі дані, зібрані за допомогою модулів, відправляються на централізований сервер, звідки можна отримати інформацію про погодні умови. Також сучасною тенденцією є розробка додатків для смартфонів, що дозволяє отримувати інформацію про погоду в мобільному режимі. Великим недоліком таких рішень є те, що в місцях використання метеостанцій необхідно проводити

інтернет, що багато в чому зменшує їх мобільність. Ще одним недоліком є обмеженість допустимого числа датчиків.

Проаналізувавши аналогічні рішення можна прийти до висновку, що основними недоліками проаналізованих платформ є:

- необхідність власноруч налаштовувати девайси та описувати код необхідний для їх функціонування та підключення до платформ, що є не бажаним для кінцевих користувачів;
- обмежена кількість готових девайсів які не потрібно налаштовувати та неможливість їх розширення;
- необхідність розміщувати платформу на власних ресурсах.

Метою даної кваліфікаційної роботи є схемотехнічна та конструкторська розобка платформи дистанційного контролю параметрів віддалених об’єктів та метеостанції.

Для досягнення поставленої мети сформульовані задачі:

- проаналізувати принципи та основні технології для роботи з велики даними;
- на основні принципів та технологій для роботи з великими даними розробити структурну схему платформи;
- на основі структурної схеми обрати компоненти для її реалізації;
- відповідно до обраної структурної схеми та компонентів розробити платформу;
- проаналізувати принципи побудування метеостанцій;
- на основні проаналізованих принципів побудування метеостанції розробити структурну схему;
- відповідно до обраної структурної схеми розробити пристрій метеостанції;
- перевірити взаємодію платформи та метостанції.

1 ОГЛЯД ТА АНАЛІЗ АНАЛОГІЧНИХ РІШЕНЬ

1.1 Огляд та аналіз платформ для управління пристроями

1.1.1 Платформа ThingsBoard

ThingsBoard – це платформа з відкритим кодом для збору, обробки даних, візуалізація та управління пристроями. Розробник описує платформу наступним чином: “Будучи надійною, масштабованою та зручною для користувачів, платформою ThingsBoard IoT підтримує різноманітні способи використання IoT, надаючи гнучкі та потужні функції «з коробки», щоб скоротити час виходу на ринок ваших підключених продуктів та розумних рішень. Платформа є пристроєм агностиком, тому ви можете подавати та аналізувати дані телеметрії з будь-якого датчика, підключеного пристрою чи програми. Комплексні функції ThingsBoard та багаті API платформи дозволяють економити час і ресурси на звичайних завданнях IoT і концентруватися на конкретних особливостях вашого рішення IoT.”[4]. Основним галузями для використання цієї платформи є:

- землеробство;
- енергетика;
- збір метрик;
- системи спостереження.

Розберемо детальніше спосіб застосування цієї платформи на прикладі збору метрик. Основними можливостями які надає дана платформа для галузі збору метрик є:

- збір даних з різних видів девайсів використовуючи різні методи синхронізації;
- візуалізація зібраних даних;
- аналіз даних;
- збереження отриманих даних для подальшого аналізу;
- оброблення даних в сторонніх додатках для подальшого обліку та

виставлення рахунків.

Типова структурна схема використання програмного забезпечення ThingsBoard для побудування рішення для збору метрик показана на рисунку 1.1.

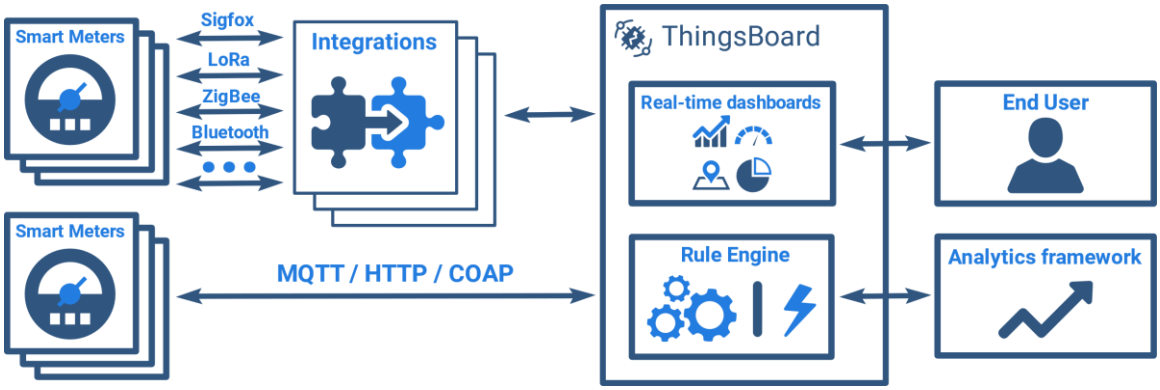


Рисунок 1.1 – структурна схема ThingsBoard

Як видно з структурної схеми платформа підтримує різні протоколи передачі інформації з віддалених пристроїв. Підтримуються як пряма передача данних через такі протоколи як MQTT, HTTP та COAP, так і передача через інтеграційні платформи такі як Sigfox, LoRa, ZigBee, Bluetooth так інші. Дані зберігаються в NoSQL базі даних – Cassandra. Також підтримується передача даних з платформи в аналітичні системи такі як – Apache Spark та Hadoop використовуючи Kafka або інші системи доставки повідомлень.

ThingsBoard надає веб додаток з панеллю моніторингу за допомогою якої можна спостерігати за станом пристроїв які наразі підключенні до системи користувача. Приклад панелі моніторингу зображено на рисунку 1.2. Ще одним важливим механізмом який надає платформа ThingsBoard є фреймворк Rule Engine (рисунок 1.3). Він дозволяє візуалізувати побудування процесу обробки подій. Завдяки фреймворку Rule Engine значно спрощується побудування рішення для обробки подій адже не потрібно описувати цей процес за допомогою коду.

Основними можливостями цього фреймворку є:

– перевірка вхідних даних;
– модифікація даних перед збереженням в базу даних;

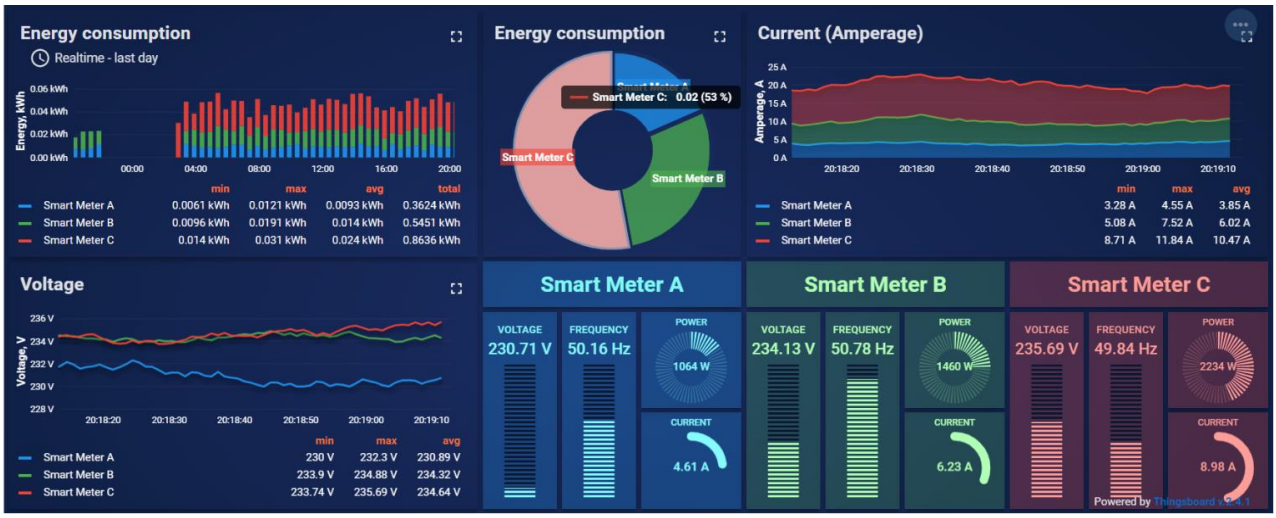


Рисунок 1.2 – Панель моніторингу ThingsBoard

– об'єднання інформації з декількох пристроїв в один актив;
– запуск дій на основі подій життєвого циклу пристрою;
– завантаження додаткових даних необхідних для обробки;
– посилання електронних листів.

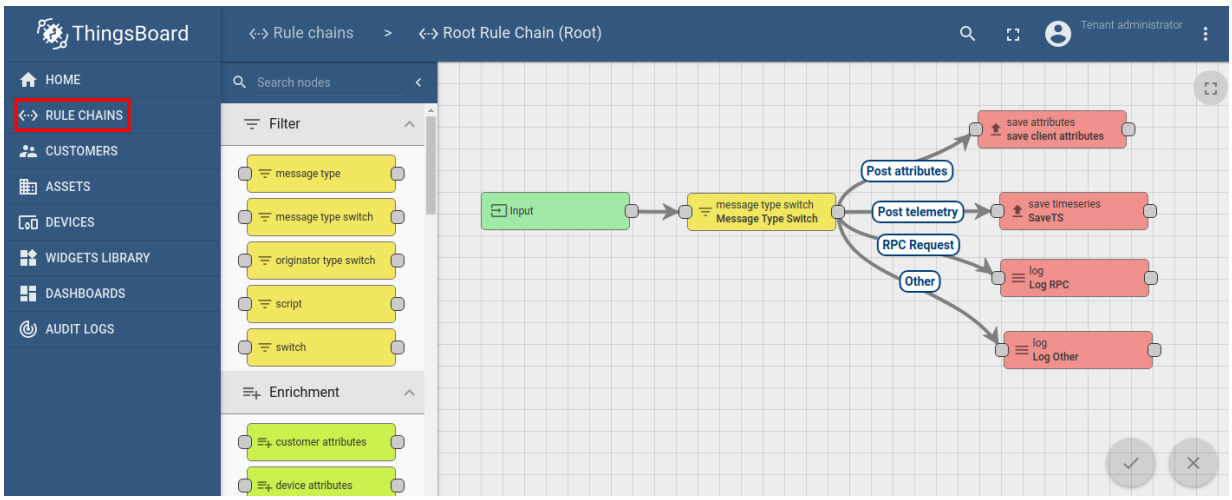


Рисунок 1.3 – Rule Engine

1.1.2 Платформа Blynk

Blynk – це IoT платформа яка дозволяє створювати свої мобільні додатки для керування пристроями. Також Blynk надає своє рішення для аналізу даних отриманих з пристроїв а також дозволяє застосувати машинне навчання за допомогою даних отриманих з пристроїв. Blynk надає власну бібліотеку яка

підтримує велику кількість моделей, наприклад, таких як ESP8266, Arduino чи Raspberry Pi. Платформа підтримує наступні протоколи передачі інформації з девайсу:

- WIFI;
- Ethernet;
- GSM, 2G, 3G, 4G, LTE;
- USB;
- Bluetooth.

Дана платформа не потребує наявності серверу або хмарного рішення адже вона інкапсулює всі операції у власному хмарному рішенні. Це значно спрощує налаштування платформи. Структурна схема платформи показана на рисунку 1.4.

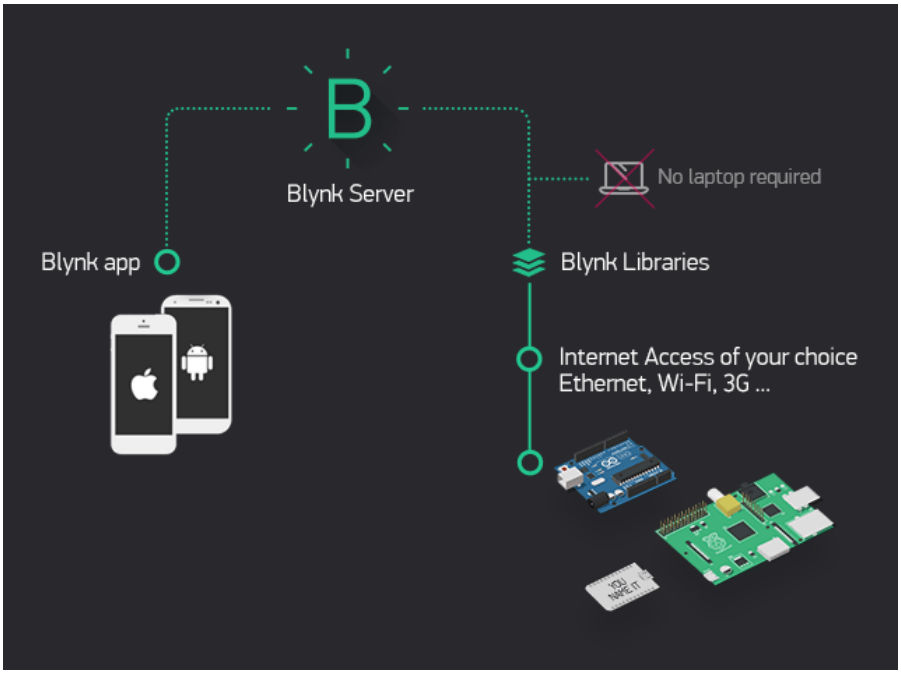


Рисунок 1.4 - структурна схема Blynk

Розглянемо спосіб роботи даної платформи:

- Для створення свого мобільного додатку потрібно скачати мобільний додаток Blynk на основі якого можна побудувати свої власні додатки. Наразі підтримуються такі платформи як IOS та Android. Інтерфейс мобільного додатку показано на рисунку 1.5.
- Далі потрібно завантажити бібліотеку яку надає платформа на прилад

який буде підключено до мережі Blynk. Це розширення, яке працює над програмним забезпеченням та обробляє всі підключення та обмін даними між приладом, Blynk Cloud та програмою яка використовується в приладі. Бібліотека підключається до програмного коду написаного для пристрою та потім завантажується на сам пристрій.

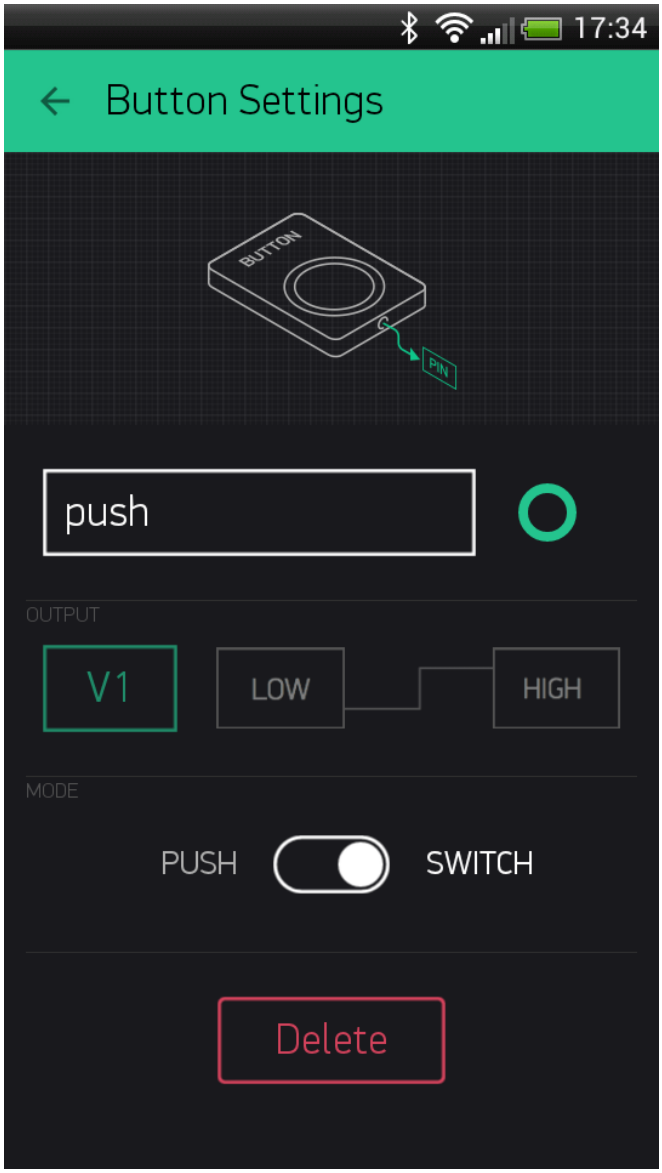
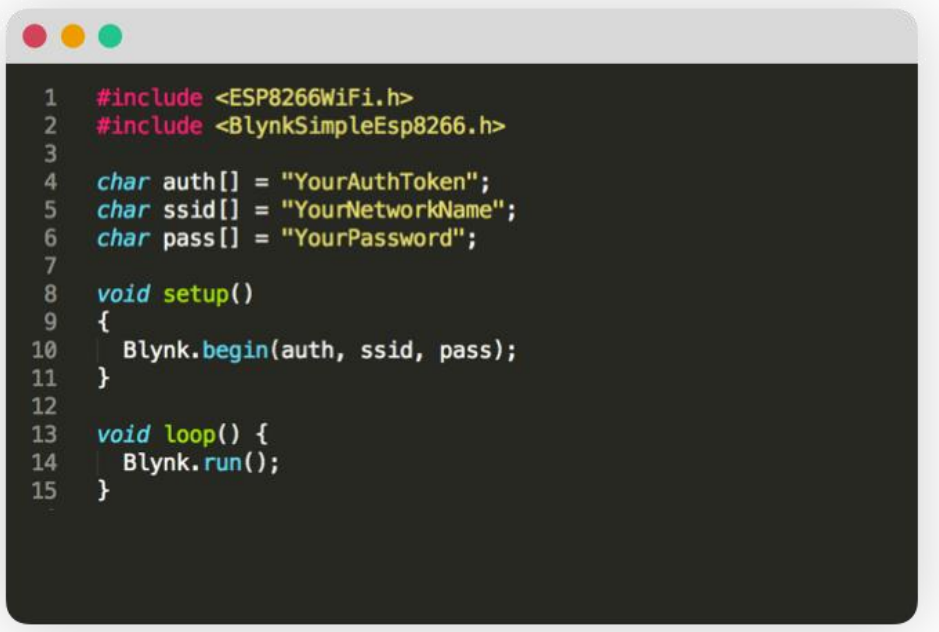


Рисунок. 1.5 – Приклад мобільного інтерфейсу

– Для того щоб підключити прилад до платформи необхідно отримати токен та додати його до коду додатку. Токен можна отримати в додатку Blynk.

– Після того як девайс був підключений до платформи необхідно додати кодову базу яка буде виконувати бажані функції. Приклад програми написаної за допомогою бібліотеки Blynk показано на рисунку 1.6.



```
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>

char auth[] = "YourAuthToken";
char ssid[] = "YourNetworkName";
char pass[] = "YourPassword";

void setup()
{
  Blynk.begin(auth, ssid, pass);
}

void loop() {
  Blynk.run();
}
```

Рисунок. 1.6 – Приклад программи з бібліотекою Blynk

Наразі платформа підтримує наступні програмні мови:

– C++;
– C#;
– Javascript;
– Python;
– MicroPython;
– HTTP;
– Node.js;
– Lua;
– OpenWrt;
– MBED;
– Node-RED.

Готовий додаток метеостанції зроблений за допомогою Blynk показано на

рисунку 1.7.

До переваг цієї платформи можна віднести велику кількість підтримуваних пристроїв, сенсорів а також протоколів передачі інформації.

Рисунок 1.7 – Приклад метеостанції

Також можна підкреслити простоту побудування свого власного

мобільного додатку, а також наявність власного хмарного рішення, що дозволяє не тратити час та ресурси на побудування свого власного рішення. Однак, основним способом використання Blynk є розробка власного додатка який буде використовуватися іншими користувачами. Тобто, платформа надає великий спектр забезпечення для розробників але не надає рішень для кінцевих користувачів.

1.1.3 Платформа myDevices

Розробник описує myDevices (рисунок 1.8) як IoT платформу, яка включає в себе апаратну, з‘єднувальну та хмарну інфраструктуру. Платформа надає спрощений спосіб для підприємців швидко та ефективно продавати або розгортати попередньо упаковані рішення, щоб збільшити свій дохід і вирішити реальні проблеми з кінцевим клієнтом. Проста, але потужна платформа, яка дозволяє продавати, керувати та підтримувати клієнтів на різних вертикальних ринках із власними фірмовими рішеннями IoT [8].

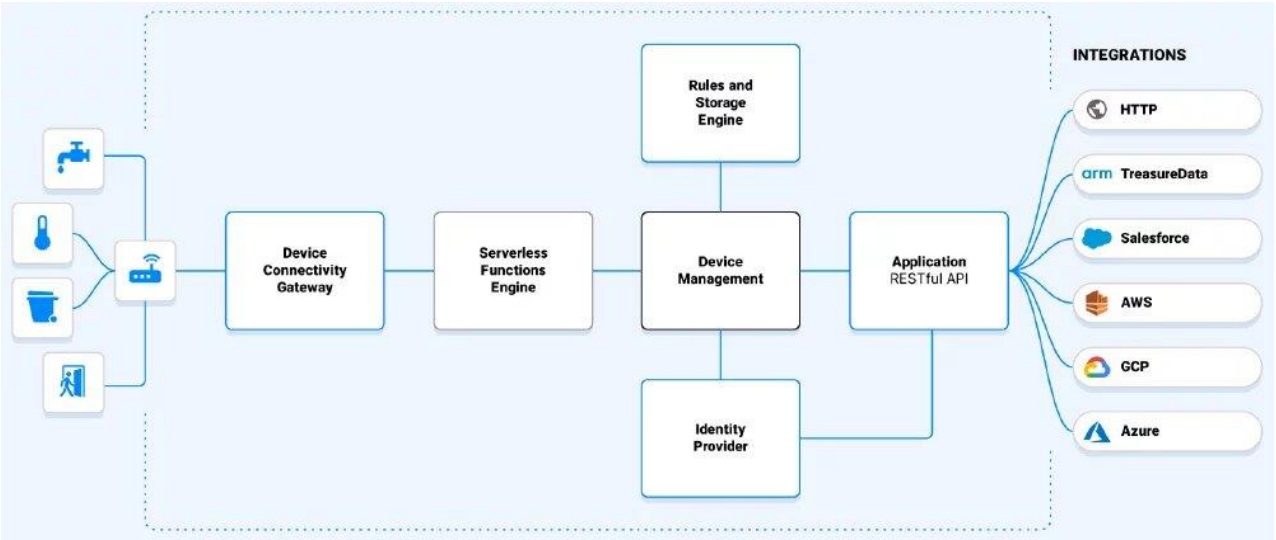


Рисунок 1.8 - Структурна схема myDevices

Розберемо детальніше основні компоненти платформи myDevices:

– Device Connectivity Gateway - Підтримує зашифровані з'єднання з пристроями, що підтримують протоколи HTTP та MQTT. Отримує трафік від мережевих серверів LoRa та потоку даних з інших хмарних рішень;

– Serverless Engine - Розшифровує та нормалізує дані пристрою висхідної лінії зв'язку та кодує команди низхідної лінії, що спрощують розгортання функцій інтеграції;

– Device Management - Дозволяє керувати реєстром пристрою, конфігурацією, забезпеченням, плануванням та групуванням FOTA. Дозволяє легко видаляти та перереєстровувати пристрої через LNS Switch. Зберігає ключі LoRaWAN та сертифікати SSL / TLS з доступом до інформації в режимі реального часу;

– Storage - Рішення для роботи з великим обсягом даних для легкого запиту мільярдів рядків телеметричних та історичних даних. Також дозволяє швидко додавати мільйони точок даних в секунду та підтримує вертикальну та горизонтальну масштабованість;

– Rules Engine - Масштабований і розширений механізм правил з розширеними налаштуваннями та конфігурацією з різними умовами та діями;

– Identity Provider - Система аутентифікації та авторизації, яка підтримує протоколи OAuth 2.0 та OpenID Connect, протоколи LDAP та Active Directory, зовнішні постачальники ідентифікаційних даних (Idp) для Google та Microsoft та SSO SAML;

– Applications - Підтримує веб та програми для iOS та Android. Також дозволяє використовувати REST API для розширення функцій програми;

– Integrartions - Дозволяє передавати данні в сторонні інтегровані програми, такі як Arm, Azure, Salesforce, Tableau та багато інших для візуалізації, аналітики звітів та машинного навчання.

Для кінцевих користувачів платформа поділяється на 2 рішення - IoT In a Box та Cayene. Розберемо детальніше кожне з них.

IoT In a Box пропонує повністю готове рішення на базі приладів з якими працює платформа. Клієнт купує пристрої або рішення яке складається з декількох пристроїв в офіційному магазині. Приклад пристрою – датчика руху показаний на рисунку 1.9. Після цього необхідно зареєструватися в платформі

та встановити мобільний додаток. За допомогою додатку просканувати QR код на придбаних пристроях. Після цього пристрої будуть автоматично додані до списку пристроїв в аккаунті. Після цього пристроями можна керувати через мобільний додаток або браузер.

Рисунок 1.9 – датчик руху

Використовуючи вже готові рішення зібрані з деяких девайсів які надає платформа можна також отримати готовий мобільний додаток створений спеціально для даного набору пристроїв. Як приклад, мобільний додаток метеостанції показаний на рисунку 1.10.

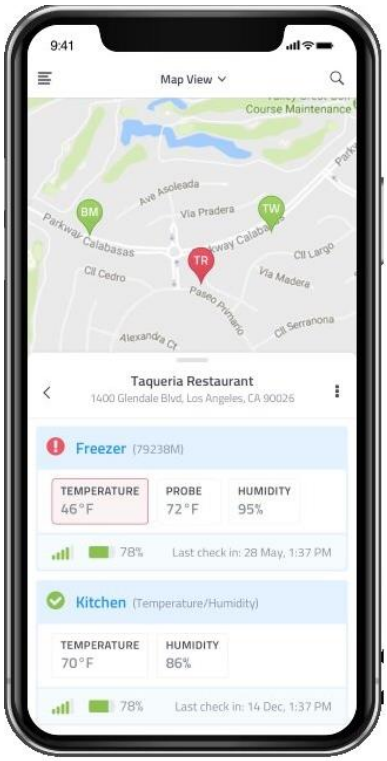


Рисунок 1.10 – Додаток метеостанції

До переваг цього рішення можна віднести мінімальну необхідну початку

настройку та наявність вже готових рішень. Але в той же час це є значним недоліком адже дозволяє користуватися тільки лімітованою кількістю пристроїв а також ускладнює налаштування мобільного чи браузерного додатку для специфічних потреб.

Cayene – це платформа яка дозволяє легко будувати рішення за допомогою Arduino або Raspberry Pi за допомогою власної бібліотеки та хмарного рішення. Панель моніторингу Cayene наведена на рисунку 1.11.

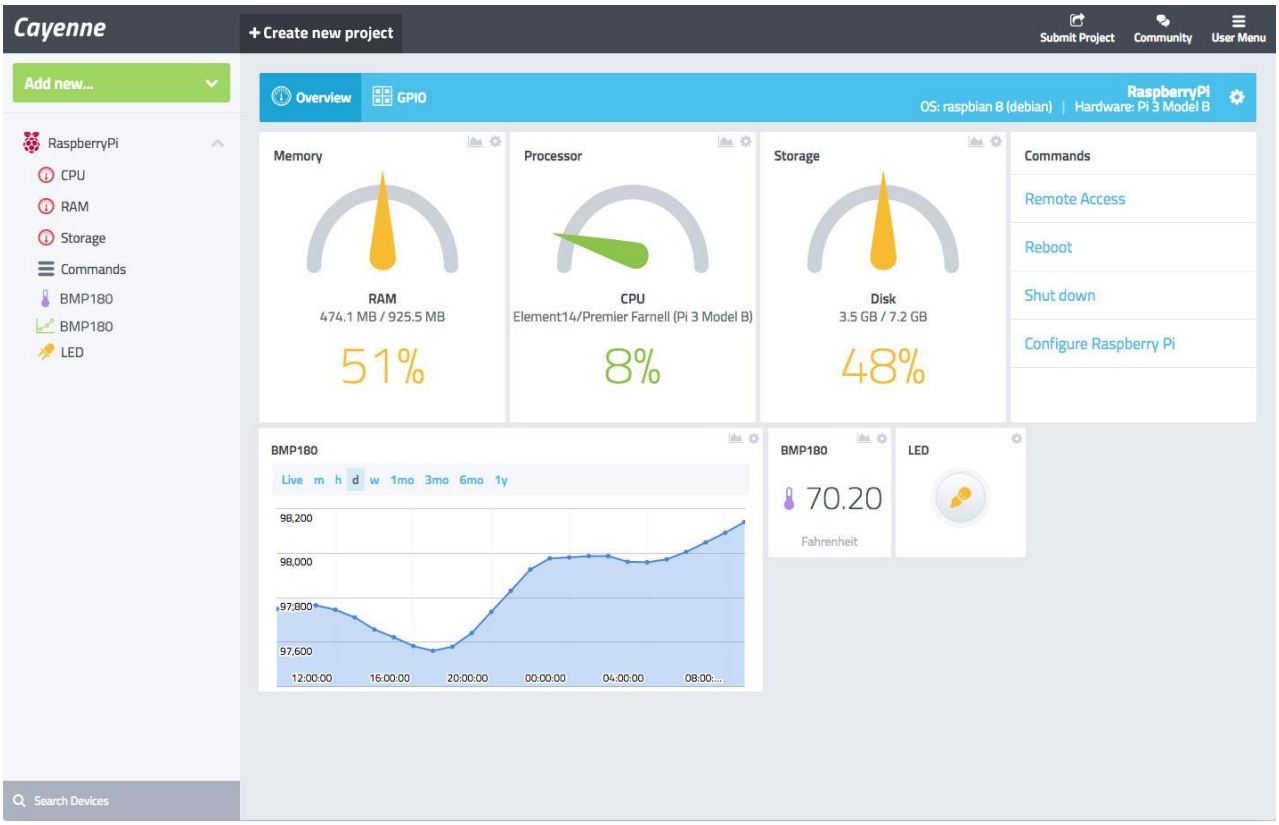


Рисунок 1.11 – Панель моніторингу Cayene

Розберемо спосіб роботи даної платформи на прикладі Arduino та діода:

– Потрібно зареєструватися в Cayene;
– Додати бібліотеку Cayene для Arduino до програмного коду;
– Після запуску девайсу він з’явиться на панелі Cayene;
– Далі необхідно підключити діод та обрати його в панелі Cayene в браузері для створення віджету;
– Після цих шагів діод буде додано до панелі девайсу в браузері;

До переваг даного рішення можна віднести велику кількість підтримуваних пристроїв та сенсорів. А також власне хмарне рішення завдяки якому не потрібно розгортати власний сервер. Це дає змогу досить швидко розробляти нові прилади з різним набором сенсорів. До недоліків можна віднести те, що дана платформа розрахована більше на розробників і не дає можливості використовувати її кінцевим користувачам як закінчене рішення.

1.2 Огляд та аналіз віддалених метеостанцій

1.2.1 Метеостанція TFA 351133 Sun

Метеостанція TFA 351133 Sun розрахована на зміну температур і властивостей повітря і дозволяє підключення до трьох зовнішніх датчиків за допомогою радіоканалу (рисунок 1.12). Максимальна відстань між датчиком і метеостанцією - 60 метрів при умові прямої видимості. Частота передачі даних 433 МГц.

Рис. 1.12 – Дистанційний датчик метеостанції TFA 351133 Sun

Метеостанція підключається до електромережі за допомогою мережевого кабелю. Також є можливість резервного живлення за допомогою 2 батарей AAA 1.5В.

У даній метеостанції не передбачена передача даних по мережі інтернет. Мобільний додаток відсутній. Вся інформація з датчиків виводиться на базовий блок. На дисплеї базового блоку (рисунок 1.13) відображається інформація про прогноз погоди на найближчі 12 годин, значення вимірюваної температури (мінімальне і максимальне), показник температури і вологість повітря.

Рисунок 1.13 – Дисплей базового блока метеостанції TFA 351133 Sun

До переваг даного пристрою можна віднести:

- простота настройки;
- ціна.

До недоліків пристрою можна віднести:

- максимальна відстань датчика не більш 60 метрів;
- відсутність можливості відправки даних через інтернет;
- відсутність мобільного додатка.

1.2.2 Метеостанція Oregon Scientific LW301

Метеостанція Oregon Scientific LW30 дозволяє вимірювати температуру,

вологість повітря, швидкість і напрям вітру, а також рівень опадів. В комплекті (рисунок 1.14) йде спеціальний модуль для підключення до інтернету, який підключається до маршрутизатора. На нього передається інформація з усіх датчиків за допомогою радіосигналу. Максимальна відстань датчиків від головного модуля - 100 метрів. Підтримується до 8 датчиків одночасно.

Рисунок 1.14 – Комплект поставки Oregon Scientific LW301

Метеостанція дозволяє вивантажувати дані на сервер і зберігати протягом одного року. Також, існує мобільний додаток для Android і IOS (рисунок 1.4). Додаток дозволяє переглядати на смартфоні в режимі реального часу інформацію про погоду, отриману від датчиків.

До переваг даного пристрою можна віднести:

- простота настройки;
- наявність мобільного додатка;
- можливість зберігати дані про погоду в хмарі.

До недоліків пристрою можна віднести:

- максимальна відстань датчика не більше 100 метрів;
- необхідність маршрутизатора і Ethernet підключення для роботи з

інтернетом;

– можливість підключення не більше восьми датчиків.

Рисунок 1.15 – Мобільний додаток метеостанції Oregon Scientific LW301

1.2.3 Метеостанція La Crosse MA10050

Метеостанція La Crosse MA10050 дозволяє вимірювати температуру і вологість повітря, швидкість і напрям вітру, а також рівень опадів. У комплект поставки входить шлюз MA10000, який дозволяє підключати до 50 різних датчиків підтримують технологію Mobile-Alerts (рисунок 1.16).

Рисунок 1.16 – Метеостанция La Crosse MA 10050

Метеостанція підключається до маршрутизатора за допомогою Ethernet кабелю і підтримує DHCP протокол для автоматичного отримання IP-адреси. Також існує мобільний додаток для Android (рисунок 1.17) і IOS в якому можна зареєструвати пристрій за допомогою QR-коду.

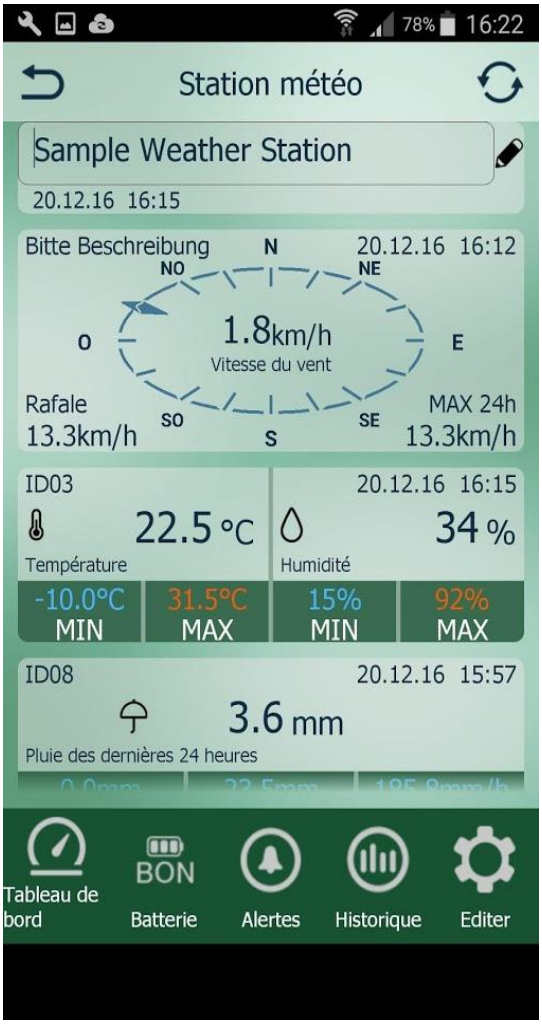


Рисунок 1.17 – Android приложение метеостанции La Crosse MA 10050

Додаток дозволяє редагувати параметри мережі мобільного шлюзу, вибирати одиниці вимірювання даних і відстежувати історію показань метеостанції за останні 90 днів з можливістю вибору дати.

А також володіє такими додатковими функціями, як настройка параметрів спрацьовування сповіщень при виході показників датчиків за встановлені значення, push-повідомлення при виході показників датчиків за встановлені значення, а також індикація низького заряду батареї пристроїв.

До переваг даного пристрою можна віднести:

– зручність настройки за допомогою DHCP протоколу;

– зручний мобільний додаток;

– наявність різних додаткових датчиків;

До недоліків пристрою можна віднести:

– необхідність маршрутизатора і Ethernet підключення для роботи з інтернетом;

– висока ціна;

– обмежена кількість датчиків для підключення.

1.2.4 Метеостанція Netatmo Weather Station

Netatmo Weather Station хоч і є домашньою метеостанцією, вона має ряд цікавих особливостей в порівнянні з раніше розглянутими метеостанціями.

Вона дозволяє вимірювати температуру повітря, рівень вологості, атмосферний тиск і рівень вуглекислого газу в повітрі і отримання звіту про його допустимій нормі.

Метеостанція Netatmo Weather Station надає мобільний додаток для Android (рисунок 1.18) і IOS . Основний модуль підключається в інтернет за допомогою WI-FI. Всі отримані дані про погоду кожні 5 хвилин вивантажуються в хмарне хранилище.

Додаток дозволяє дивитися прогноз погоди, погодні схеми і цикли, рівень вологості і вуглекислого газу.

До переваг даного пристрою можна віднести:

– зручний мобільний додаток;

– підключення до інтернету через WI-FI;

– можливість зберігати дані в хмарі.

До недоліків пристрою можна віднести:

– обмежена кількість датчиків для підключення;

– необхідність роутера для підключення до інтернету.

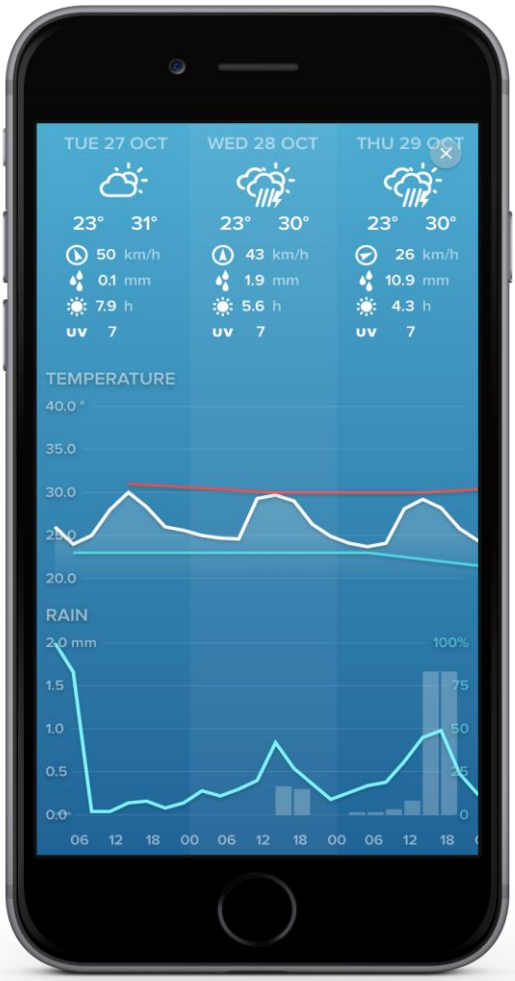


Рисунок 1.18 – Мобільний додаток метеостанції Netatmo Weather Station

1.3 Висновки

Проаналізувавши аналогічні рішення можна прийти до висновку, що основними недоліками проаналізованих платформ є:

- необхідність власноруч налаштовувати девайси та описувати код необхідний для їх функціонування та підключення до платформ, що є не бажаним для кінцевих користувачів;
- обмежена кількість готових девайсів які не потрібно налаштовувати та неможливість їх розширення;
- необхідність розміщувати платформу на власних ресурсах.

Беручи до уваги недоліки описані вище в даній роботі планується розробити

платформу яка буде надавати можливість легко розробляти нові девайси для розробників а також легко підключати девайси кінцевим користувачам без необхідності зміни кодовій бази девайсу а також надавати можливість слідкувати за показниками які надходять з девайсів.

2 ВЕЛИКІ ДАННІ

2.1 Огляд

Великі данні – це колекція даних дуже великих розмірів яка до того ж постійно зростає. Як правило данні настільки великі, що традиційні способи зберігання та обробки даних не є ефективними. Тому при обробці та зберіганні великих даних використовують технології та практики які об’єднуються під назвою великі данні. Великі данні можуть зберігатися в одному з 3 форматів:

– структуровані данні – це данні які зберігаються в одному з структурованих форматів, наприклад у реляційній базі даних. Це дозволяє тримати всі данні в одному форматі, що спрощує роботу з ними, але якщо розмір даних стає дуже великим то зберігання даних у структурованому виді може призвести до того, що вони не будуть корректно працювати.

– неструктуровані данні – це данні які зберігаються у будь–якій формі, що призводить до того що їх важко обробляти адже данні можуть бути у різних формах. Однак перевагою такого типу даних є можливість зберігання їх у дуже великому розмірі. Типовими прикладами не структурованих даних є данні які містять в собі комбінацію тексту, зображень, відеозаписів тощо.

– напів–структуровані данні – це данні які можуть містити в собі як структуровані так і не структуровані данні. Прикладом таких даних є XML або JSON файли.

Основними властивостями якими визначаються великі данні є:

– обсяг – сам термін великі данні пов’язаний з розміром. Розмір даних відіграє велику роль у визначення важливості даних. Велику кількість даних не можна зберігати на одній машині, тому одним з питань які вирішують великі данні це можливість збереження великої кількості даних на різних машинах;

– різноманітність – різноманітність стосується неоднорідних джерел та характеру даних як структурованих так і неструктурованих. Якщо раніше

основним джерелом даних були електронні таблиці, то зараз ми маємо справу з електронним листами, фотографіями, відеозаписами, пристроями моніторингу, PDF–файлами, аудіо записами тощо. Це різноманіття даних потребує певних методів для їх обробки.

– швидкість – данні з різних систем як правило приходять постійно та безперервному. Це призводить до необхідності побудування рішень які зможуть швидку обробляти такі потоки даних.

– змінність – з часом формат даних які надходять від одного джерела можуть змінюватися, що викликає необхідність у створенні механізмів які зможуть обробляти такі випадки.

Існує дві основні стратегії обробки великих даних – пакетна та потокова. Пакетна обробка використовується коли необхідна обробити великий обсяг різноманітних даних. Потокова обробка використовується у випадку коли необхідно миттєва реакція на данні які надходять.

Під час роботи з великою кількістю даних виникає проблема нехватки ресурсів. Існує два основних варіанти вирішення цієї проблеми:

– вертикальне масштабування;
– горизонтальне масштабування.

При використанні вертикального масштабування до машини додаються додаткові ресурси для підвищення обчислювальних потужності (наприклад, процесор чи оперативна пам’ять). Вертикальним масштабуванням легше керувати в порівнянні з горизонтальним і воно є дуже ефективним якщо працювати з даними порівняно невеликого розміру. При необхідності обробляти великі об’єми даних використовується горизонтальне масштабування. Замість додавання додаткових ресурсів до машини, натомість додаються нові машини та обробка даних розподіляється поміж всі доступних машин.

Одним з основних принципів які використовується при обробці великих даних на великій кількості машин є MapReduce. MapReduce – це програмна

модель та програмний каркас, що її реалізує, розроблені компанією Google для проведення розподіленої паралельної обробки великих масивів даних з використанням кластерів звичайних недорогих комп'ютерів. Принцип роботи MapReduce показаний на рисунку 2.1. Програма MapReduce складається із функції Map(), яка обробляє пари ключ/значення і генерує набір проміжних пар ключ/значення, і функції Reduce(), яка зводить докупи всі проміжні значення пов'язані з одним і тим же проміжним ключем [61].

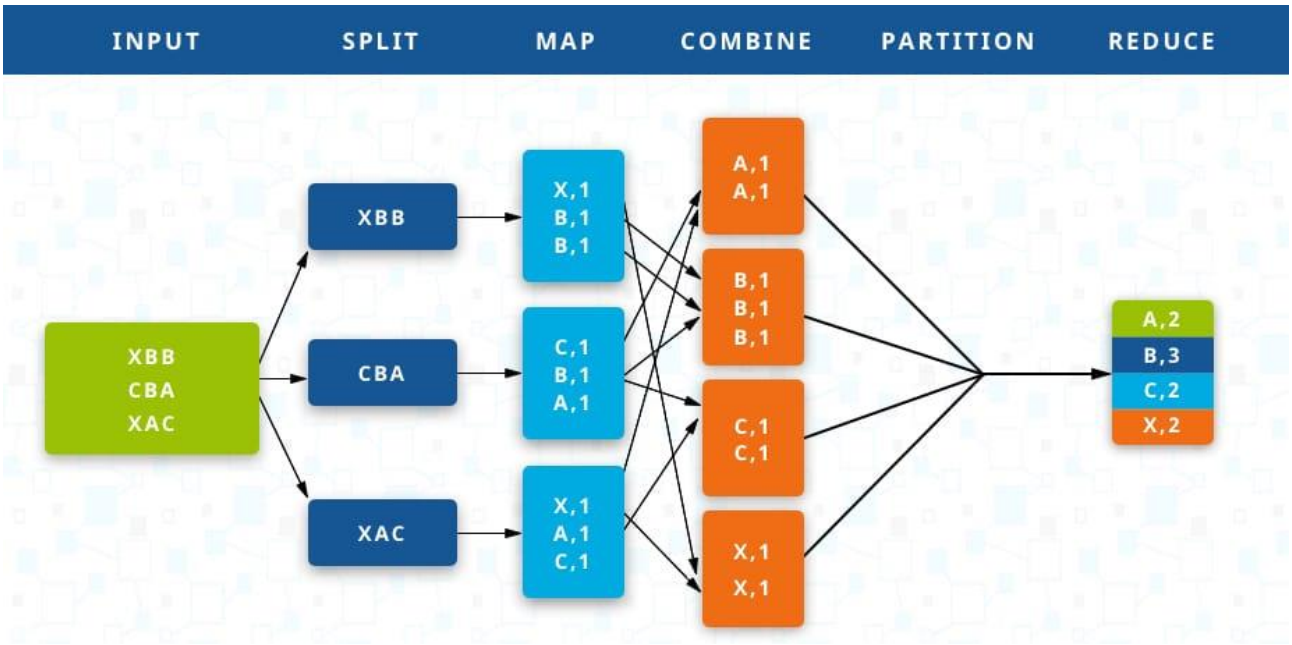


Рис. 2.1 – Принцип роботи MapReduce

При використанні MapReduce вхідні данні розподіляються на різні частини які надсилаються на різні машини для обробки а потім агрегуються в кінцевий результат.

2.2 Apache Spark

Apache Spark є єдиним механізмом для масштабованої обробки даних [41]. Основними перевагами якими володіє Apache Spark є:

– сервіс досягає високої продуктивності як для пакетних так і для потокових рішень завдяки використанню оптимізатора запитів, механізму

фізичного виконання та завдяки тому, шо проміжні результати не записуються на диск а зберігаються лише у RAM пам’яті;

– сервіс надає понад 80 операторів високого рівня, які спрощують створення паралельних додатків. Ці оператори можуть використовуватися інтерактивна за допомогою бібліотек для таких мов програмування як – Scala, Python, R та SQL;

– сервіс може працювати як у власному дата–центрі так і в хмарних сервісах.

Зазвичай Apache Spark використовується для:

– читання та збереження даних у режимі реального часу;

– обробки великої кількості данних за допомого SQL запитів;

– аналізу данних використовуючі машинне навчання та нейронні мережі.

Для досягнення заявлених можливостей Apache Spark надає набір модулів (рисунок 2.2) які дозволяють вирішувати різні задачі а поєднуючи їх між собою надають можливість вирішувати складні проблеми при роботі з великими даними.

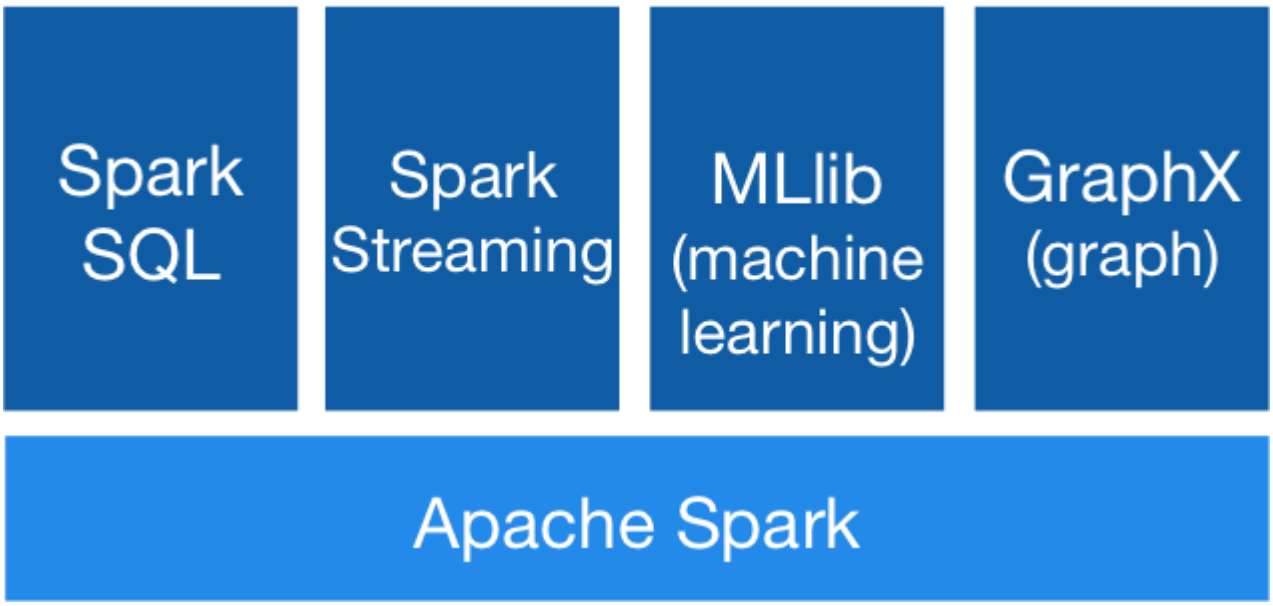


Рисунок 2.2 – Основні модулі Apache Spark

Розглянемо кожну з бібліотек детальніше:

– Spark SQL – це модуль для роботи з структурованими даними. Модуль дозволяє будувати запити до різних типів ресурсів, таких як Hive, Avro, Parquet, ORC, JSON, та JDBC;

– Spark Streaming – це модуль для побудування потокових додатків. Модуль підтримує відмовостійкість. Це дозволяє відновлювати втрачену роботу та стан оператора без необхідності написання додаткового коду;

– MLlib – це масштабована бібліотека для машинного навчання. Завдяки використанню високоякісних алгоритмів та використанню ітеративних обчислень MLlib може давати кращі результати ніж підходити які використовуються у MapReduce;

– GraphX – це модуль для роботи з графовими та паралельно–графовими обчисленнями. Дозволяє працювати як з графами так і з колекціями. GraphX дозволяє виконувати аналіз та ітераційні обчислювання. Також GraphX містить в собі велику кількість алгоритмів які можна використовувати для роботи з графами.

Під час використання Apache Spark данні паралізуються за допомогою стійких розподілених наборів даних(RDD). RDD – це базова абстракція Apache Spark яка використовує вхідні данні для створення розділів які розподіляються на різні кластери. RDD використовується для виконання операцій трансформації (Transformations) та дії (Actions) які створюють кінцевий результат (Result) (рисунок 2.3).

Трансформації використовуються для створення нового представлення даних з вхідного RDD за допомогою різних операцій. Нове представлення даних теж знаходиться в форматі RDD, що дозволяє виконувати операції для зміни представлення даних декілька разів під час однієї обробки. Всі трансформації є лінивими, що означає що вони виконуються тільки один раз під час виклику. Дії використовуються для отримання результатів аналізу з Apache Spark до програми яка працює з платформою. RDD працюють, розбиваючи свої данні на ряд розділів. Розділи RDD потрібно зберігати на

кожному з вузлів виконавця. Розміщення розділів на кожному з вузлів виконавця дозволяє у випадку якщо один з вузлів стає недоступним, відновити втрачені розділи. Також Apache Spark дозволяє зберігати пари ключ/значення. Для цього Apache Spark використовує парні RDD. Парні RDD формуються за допомогою двох RDD які зберігаються в кортежі. Перший RDD використовується для збереження ключів, а другий для збереження значень.

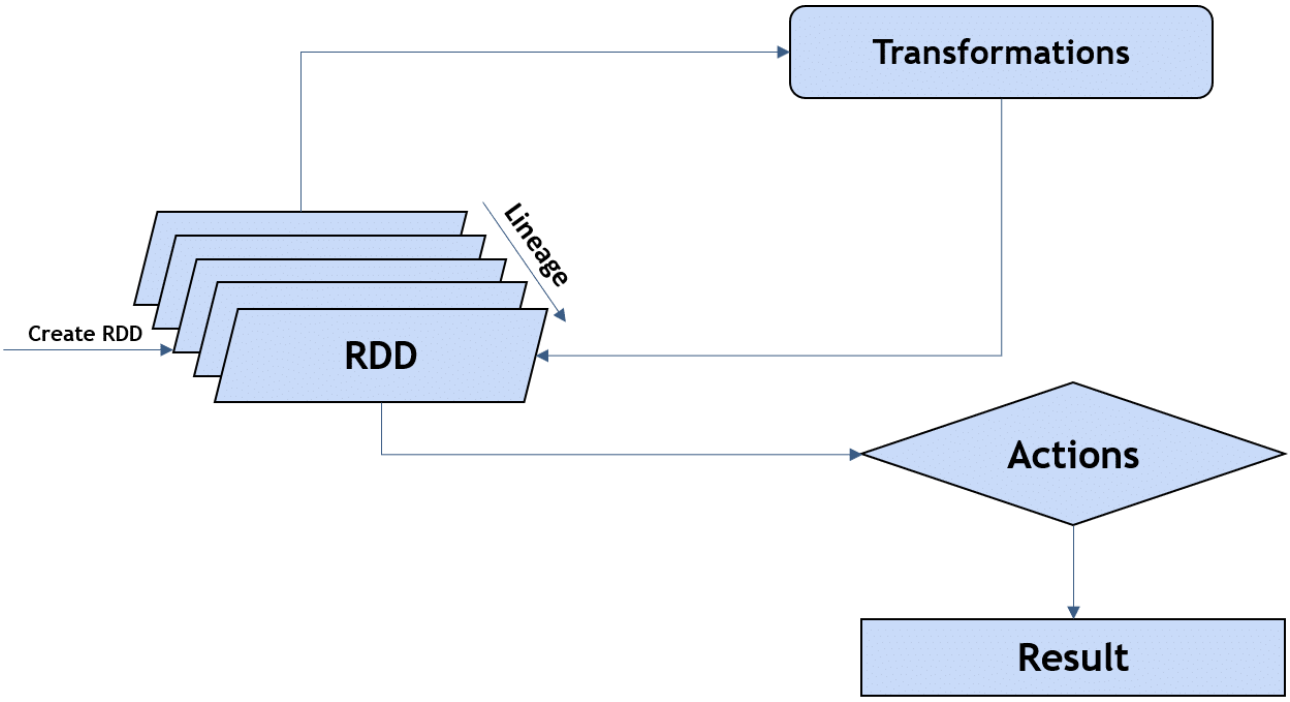


Рисунок 2.3 – Приклад роботи з RDD

2.3 ETL

ETL (абревіатура від Extract (вилучення), Transform (перетворення) та Load (завантаження) – це процес вилучення даних із вхідної бази даних, перетворення та форматування їх для потреб системи, яка їх споживає, та завантаження їх у базу даних [42]. ETL використовують для вирішення різних задач серед яких:

- консолідація, збір та приєднання даних від зовнішніх постачальників
- переміщення даних із застарілих систем до нових систем з різними

форматами даних

- очищення, профілювання та аудити даних
- валідації даних

Схема процесу роботи процесу ETL наведена на рисунку 2.4.

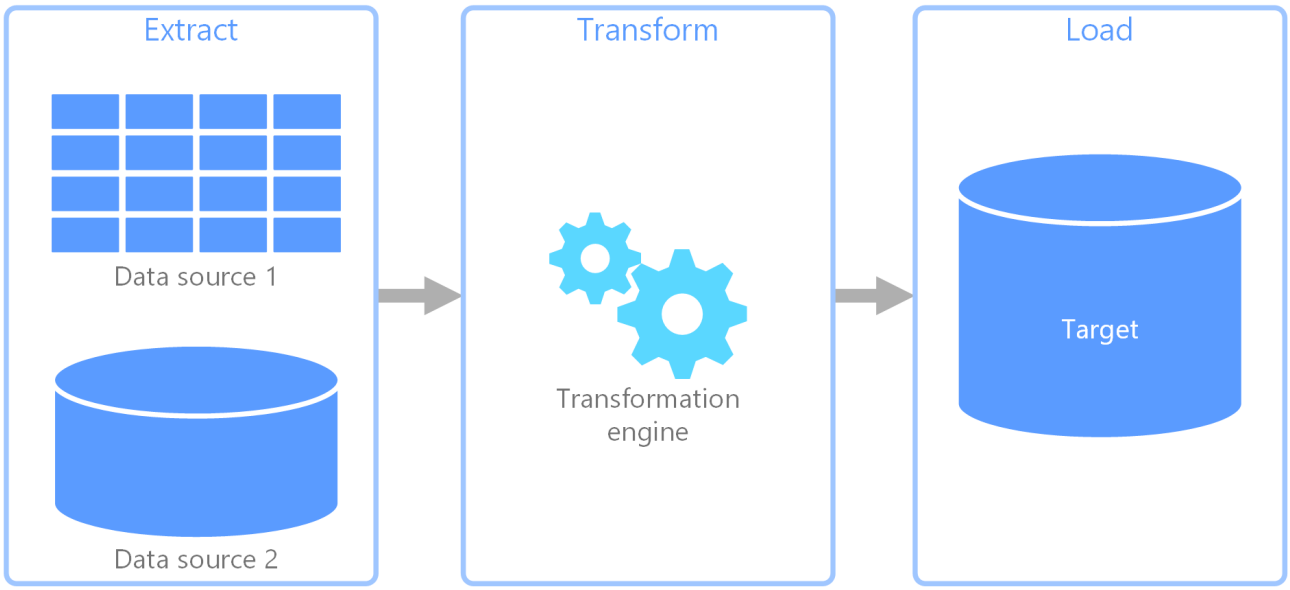


Рисунок 2.4 – Схема роботи процесу ETL

Існує декілька варіантів використання технології ETL – Batch(пакетна) та Stream(потокова). Batch ETL використовується у випадку коли необхідно обробляти велику кількість даних в встановлений час. Як правило пакетна обробка використовується під час періоду коли доступні обчислювальні ресурси мають низьку загрузку. В порівнянні з потоковою обробкою, пакетна обробка є більш простою системою, яка не потребує спеціального обладнання або системної підтримки для додавання нових даних. Після налаштування пакетна обробка вимагає менше обслуговування, ніж потокова. Можливість для помилок зменшується під час використання пакетної обробки оскільки пакетна обробка автоматизує більшість або всі компоненти та мінімізує взаємодію з користувачем. Також пакетна обробка прискорює обробку даних в цілому адже вона дозволяє обробляти данні одночасно. Потокова обробка – це метод обробки даних, в якому інформація аналізується та організовується під час її генерування. Після отримання даних про подію вона відразу обробляється та

доставляється до місця призначення. Потокова обробка даних спрощує та пришвидшує аналіз даних, щоб забезпечити швидкий доступ до інформації. Порівнюючи два види обробки даних слід зазначити, що пакетну обробку даних використовують для дуже великих обсягів даних або даних з застарілих систем, які не здатні забезпечити потокову передачу даних. Пакетна обробка також, зазвичай, використвоується коли час не є критичним фактором. Пакетна обробка даних є найбільш корисною, коли обробка великих обсягів даних важливіша, ніж швидке отримання результатів. Натомість, потокову обробку даних краще використовувати коли бажані майже миттєві результати. Наприклад, у сфері IoT де інформація з датчиків надходить постійно, користувач хоче бачити нову інформацію відразу і реагувати коли це необхідно більш підходить потокова обробка даних, адже вона дозволяє майже миттєво надавати нові результати з датчиків.

Оскільки сьогодні зростає популярність безсерверних обчислень, вони починають використовуватися і у процесі ETL. Використання бесерверних обчислень дозволяє спростити побудування рішення за допомогою ETL адже дозволяє не витрачати час на виділення ресурсів, їх конфігурування та підтримку. Також це дозволяє виконувати процеси ETL у більш масштабованому середовищі, що дозволяє не турбуватися про можливості пропускної здатності.

2.4 Hadoop

Hadoop – це каркас, який дозволяє розподіляти обробку великих наборів даних по кластерах комп'ютерів за допомогою простих моделей програмування. Він розроблений для масштабування від одного сервера до тисяч машин, кожен з яких пропонує локальні обчислення та зберігання. Замість того, щоб покластися на апаратне забезпечення для забезпечення високої доступності, сама бібліотека призначена для виявлення та обробки помилок системи на рівні

додатків, тому надає високодоступний сервіс поверх кластера комп'ютерів, кожен з яких може бути схильним до збоїв [44]. Hadoop це сучасне рішення яке використовується для різних задач. Наразі основними компонентами Hadoop є:

– YARN;
– HDFS.

2.4.1 YARN

Hadoop YARN – компонент який керує та планує ресурси системи, розділяючи навантаження на кластери машин. YARN відповідає за розподіл системних ресурсів для різних додатків що працюють в кластері Hadoop, і планування завдань, що виконуються на різних вузлах кластера. В архітектурі кластерів Apache Hadoop YARN розташований між HDFS та процесорами, що використовуються для запуску програм. Він поєднує центральний менеджер ресурсів з контейнерами, координаторами додатків та агентами рівня вузлів, які відстежують операції обробки в окремих вузлах кластера.

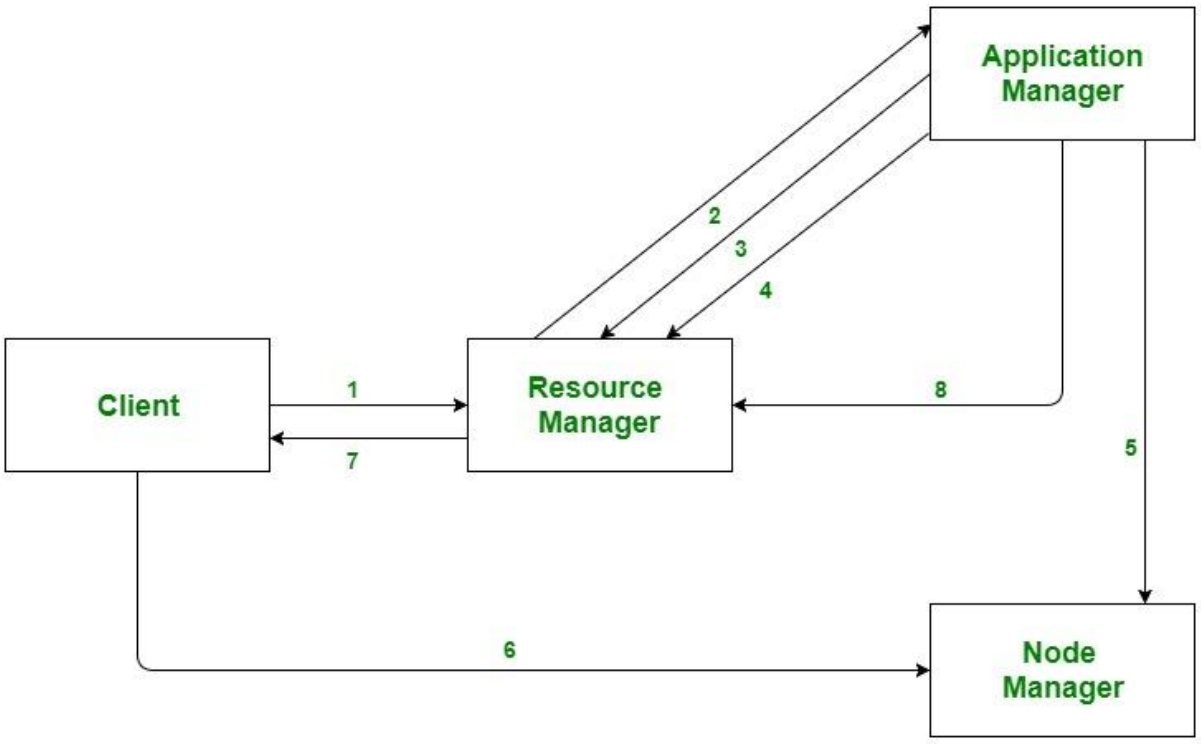


Рисунок 2.5 – Принцип роботи Hadoop YARN

YARN може динамічно розподіляти ресурси для додатків. Крім того,

YARN підтримує кілька методів планування, всі засновані на форматі черги. Hadoop YARN використовує планувальник який розподіляє доступні ресурси між задачами виходячи з кількості необхідних ресурсів для кожної задачі. Hadoop YARN також включає функцію системи резервування, яка дозволяє користувачам заздалегідь резервувати ресурси кластера для важливих завдань обробки, щоб забезпечити їх безперебійне виконання. Apache Hadoop має наступні компоненти для розподілу ресурсів між задачами:

– ResourceManager, який приймає подані завдання від користувачів, планує завдання та розподіляє їм ресурси;

– NodeManager, який встановлюється на кожному з вузлів і функціонує як агент моніторингу та відправляє данні в ResourceManager;

– ApplicationMaster, створений для кожної програми для узгодження ресурсів та роботи з NodeManager;

– Контейнери ресурсів, які контролюються за допомогою NodeManagers.

На рисунку 2.5 показаний принцип роботи Hadoop YARN який складається з наступних шагів:

– клієнт подає заявку;

– менеджер ресурсів виділяє контейнер для запуску диспетчера додатків;

– менеджер програм реєструється в диспетчері ресурсів;

– менеджер програм узгоджує контейнери з менеджером ресурсів;

– менеджер прикладних програм повідомляє диспетчеру вузлів про запуск контейнерів;

– код програми виконується в контейнері;

– клієнтські отримує інформацію про стан виконання програми з менеджеру ресурсів;

– після завершення обробки менеджер програм повідомляє менеджеру ресурсів про завершення виконання программи.

Використання Apache Hadoop YARN для відділення HDFS від MapReduce зробило середовище Hadoop більш придатним для використання в режимі реального часу та для використання інших програм, які не можуть чекати завершення пакетних завдань.

2.4.2 HDFS

HDFS (Hadoop Distributed File System) – це розподілена файлова система. Вона має багато подібності в порівнянні з існуючими розподіленими файловими системами. Однак відмінності від інших розподілених файлових систем значні. HDFS відрізняється високою стійкістю до відмов і розроблена для використання на апаратному забезпеченні з низькими витратами. HDFS забезпечує високу пропускну здатність для даних додатків і підходить для програм, що мають великі набори даних. HDFS послаблює кілька вимог POSIX, щоб забезпечити потоковий доступ до даних файлової системи. HDFS був спочатку побудований як інфраструктура для проекту веб–пошукової системи Apache Nutch [46]. Основними цілями які ставить перед собою HDFS є:

– виявлення системних проблем та відновлення. Оскільки HDFS включає велику кількість обладнання, вихід з ладу компонентів є частим явищем. Тому HDFS надає механізми швидкого та автоматичного виявлення проблем та відновлення.

– HDFS має мати сотні вузлів на кластері для управління програмами, що мають великі набори даних.

– ефективне виконання завдань над даними

Архітектура HDFS наведена на рисунку 2.6. Кластер HDFS складається з одного головного сервера (NameNode), який управляє простором імен файлової системи та регулює доступ клієнтів до файлів. Крім того, існує ряд серверів даних (DataNodes), як правило, один сервер на вузол кластера, які управляють сховищем, приєднаним до вузлів, на яких вони працюють. HDFS відкриває простір імен файлової системи та дозволяє зберігати дані користувачів у файлах. Всередині файл розділений на один або кілька блоків, і ці блоки

зберігаються у наборі вузлів даних. NameNode виконує такі операції як відкриття, закриття та перейменування файлів і каталогів в просторі імен файлової системи. Він також визначає відображення блоків на DataNodes. DataNodes відповідають за обслуговування запитів для читання та запису від

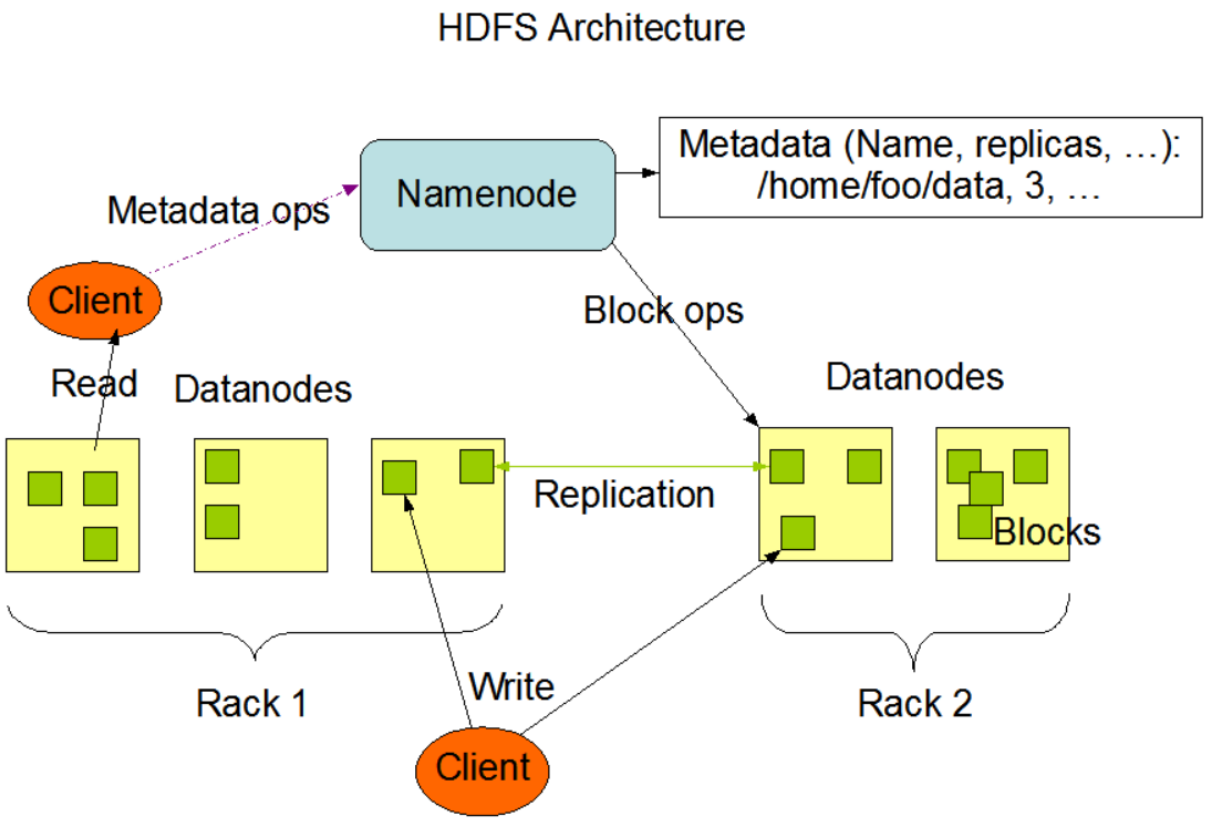


Рисунок 2.6 – Архітектура HDFS

клієнтів файлової системи. DataNodes також виконують створення, видалення та реплікацію блоків за інструкцією з NameNode.

HDFS побудований з використанням мови Java; будь–яка машина, яка підтримує Java, може запускати NameNode або DataNode. Використання високо портативної мови Java означає, що HDFS можна розгорнути на широкому діапазоні машин. . Існування єдиного NameNode в кластері значно спрощує архітектуру системи. NameNode є арбітром і сховищем для всіх метаданих HDFS. Система створена таким чином, що дані користувача ніколи не протікають через NameNode.

2.5 Зберігання та обробка великих даних в платформі

Проаналізувавши основні підходи та бібліотеки для роботи з великими даними, стає зрозумілою необхідність підтримки платформою для дистанційного контролю параметрів віддалених об’єктів зберігання та обробки великих даних. Беручи до уваги той факт, що данні з девайсів надходять постійно, стає зрозумілою необхідність використовувати підхід потокової обробки даних, який забезпечить швидке надходження даних з сенсорів до користувачів, що дозволить останні отримувати інформацію майже у режимі реального часу. Так як для підходу потокової обробки даних характерне постійне отримання нових даних, необхідно забезпечити механізм завдяки яку платформа буде надавати велику пропускну спосібність для даних які до неї надходять, тим сам мінімізуючи можливість втрати даних що надходять. Також необхідно обрати рішення для зберігання даних яке буде дозволяти зберігати в собі велику кількість даних, обробляти їх та легко масштабуватися. Окрім того, необхідно забезпечити легке інтегрування одного з рішень для роботи з великими даними. Передбачення можливості легкого інтегрування дозволить у майбутньому додати можливість виконувати додаткові аналітичні та перетворювальні операції над даними з девайсів, що дозволить користувачам отримувати додаткову аналітику для всіх своїх девайсів а також інші представлення своїх даних.

2.7 Висновки

Проаналізувавши методи роботи з великими даним та основні технології можна прийти до висновку, що платформа дистанційного контролю параметрів віддалених об’єктів повинна бути побудувана дотримучь основних приципів роботи з великими даними, таких як ETL та повинна забезпечувати масштабованість для обробки великих обсягів інформації яка надходить з девайсів.

3 РОЗРОБКА АРХІТЕКТУРИ ПЛАТФОРМИ

3.1 Огляд основного фунціоналу платформи

Беручи до уваги задачу яка була поставлена розглянемо основний функціонал який повинна мати платформа. Основними вимогами є:

– платформа повинна мати можливість отримувати данні з девайсів, зберігати їх, та показувати їх користувачам;

– данні з девайсів повинні бути захищенні;

– платформа повинна надавати зручний програмний інтерфейс за допомогою якого користувачі зможуть з нею взаємодіяти;

– платформа повинна бути розміщена на власних ресурсах та не потребувати від користувача розміщення на його власних.

Виконання цих умов дозволить створити платформу яка дозволить користувачам використовувати вже існуючи або розробляти нові девайси яким для підключення до платформи потрібен лише доступ до інтернету. Після підключення девайсу до платформи користувач буде мати змогу спостерігати за показниками які надходять з девайсів а також управляти своїми пристроями або додавати нові.

3.2 Огляд можливих проблем та їх рішень

3.2.1 Передача великої кількості даних з датчиків

Так як планується що платформа буде єдиної для великої кількості датчиків які будуть до неї підключатися існує проблема забезпечення необхідної пропускної здатності для доставки повідомлень з девайсів до платформи, адже якщо брати до уваги той факт, що як правило девайси доволі часто збирають корисну інформацію яка повинна передаватися до платформи, то маючи велику кількість девайсів які надсилають велику кількість

повідомлень ми маємо ситуацію постійного великого навантаження яку платформа повинна витримувати.

Існує велика кількість рішень для забезпечення великої пропускної здатності. Так, наприклад, одними з найпопулярніших є використання горизонтально масштабування. Горизонтальне масштабування це термін який описує можливість системи справлятися з великою загрузкою за допомогою додавання нових ресурсів. На відмінну від вертикального масштабування, рішення яке вирішує питання боротьби з великою навантаженням за допомогою збільшення ресурсів обчислювальних машин, горизонтальне масштабування, натомість, додає нові обчислювальні машини, що надає змогу більш гнучкого масштабування, адже додавати нові машини набагато легше аніж постійно змінювати ресурсні конфігурації вже існуючих машин.

Ще однією практикою боротьби з великою пропускною спосібністю є використання черг. Черга – це динамічна структура даних, що працює за принципом «перший прийшов — перший пішов» (англ. FIFO — first in, first out). У черги є голова (англ. head) та хвіст (англ. tail). Елемент, що додається до черги, опиняється в її хвості. Елемент, що видаляється з черги, знаходиться в її голові [23]. В даному випадку черги можуть використовуватися як механізм для зберігання даних які надходять до девайсів до того моменту як вони будуть обробленні. Це дозволяє знизити кількість ресурсів необхідних для сервісів які будуть обробляти данні, адже в випадку якщо записувати данні напряму з девайсів в сервіси то необхідно забезпечувати постійну можливість сервісів обробляти всі данні які надходять, тому що у разі якщо на момент приходу повідомлення з девайсу всі сервіси буду зайняті обробкою повідомлень які прийшли до цього, то повідомлення буде втрачено. У разі ж надсилання повідомлень в чергу, сервіси здатні вичитувати з неї повідомлення коли у них звільняються на це ресурси. Цей механізм дозволяє з одного боку уникнути ситуації коли повідомлення буде втрачено, а з іншого дозволяє знизити кількість ресурсів необхідних сервісам обробникам повідомлень.

Комбінуючи горизонтальне масштабування та черги можна досягти

бажаного результату, а саме зростання ресурсів для обробки інформації яка приходить з датчиків у разі якщо кількість інформації в один проміжок часу буде зростати, а також зменшити навантаження на ресурси за допомогою впровадження проміжного сховища.

3.2.2 Зберігання даних

Як було зазначено раніше з датчиків може надходити велика кількість даних. Враховуючи що данні необхідно зберігати для того щоб показувати їх користувачам або для подальшої обробки та аналізу необхідно обрати сховище в якому данні будуть зберігатися. Основним критерієм при виборі сховища є підтримка сховищем великих даних.

Загалом, велика кількість баз даних підтримує масштабування та може бути розглянута, але при виборі бази даних також необхідного враховувати зусилля які необхідні для зберігання великих даних. Так, як правило, реляційні бази даних здебільшого мають проблеми з горизонтальним масштабуванням, а не реляційні бази даних, хоч і мають більш зручні механізми для масштабування, не підтримують реляційну модель даних, що може бути критичним. Компромісом у даній ситуації є використання хмарних рішень які надають повністю автоматизовані сервіси баз даних та вирішують проблеми масштабування.

3.2.3 Доступ користувачів до даних отриманих з девайсів

Так як користувачі не можуть отримати доступ до бази даних в якій зберігаються данні отримані з девайсів з поглядів безпеки існує необхідність в побудуванні додаткового рішення яке стане прошарком між базою даних та кінцевими користувачами та буде надавати зручний контракт доступу для останніх. Як правило, для подібних цілей використовується веб сервіс який має доступ до бази даних, а також надає можливість отримувати з неї данні за допомогою відправлення запитів по протоколу HTTP. HTTP – це протокол передачі даних, що використовується в комп'ютерних мережах. Назва

скорочена від Hyper Text Transfer Protocol, протокол передачі гіпер-текстових документів [24]. Такий підхід дозволяє ізолювати базу даних від користувачів а також надати зручний контракт за допомогою якого можна зручно побудувати веб чи мобільний додаток або отримувати данні в іншому сервісі.

3.2.4 Передача інформації з девайсу до платформи

Ще однією проблемою яку необхідно вирішити є визначення протоколу за допомогою якого данні будуть надсилатися з девайсу до платформи. Наразі, основними протоколами є:

- HTTP;
- MQTT.

Протокол HTTP використовується для клієнт-серверної комунікації, що означає що клієнт сам ініціює запит до серверу. HTTP був розроблений ще у 1990-х роках але й наразі є основним протоколом передачі інформації у вебі. Кожен запит відправляється на сервер, який обробляє його та повертає відповідь. Між запитами та відповідями як правило існує велика кількість посередників, які називаються проксі [25].

Проксі – це сервер (комп'ютерна система або програма) в комп'ютерних мережах, що дозволяє клієнтам виконувати непрямі (через посередництво проксі-сервера) запити до мережевих сервісів. Спочатку клієнт з'єднується з проксі-сервером і запитує який-небудь ресурс (наприклад, e-mail), розташований на іншому сервері. Потім проксі-сервер або підключається до вказаного сервера і отримує ресурс у нього, або повертає ресурс з власного кешу (у випадках, якщо проксі має свій кеш). У деяких випадках запит клієнта або відповідь сервера може бути змінена проксі-сервером з певною метою. Також проксі-сервер дозволяє захищати клієнтський комп'ютер від деяких мережевих атак і допомагає зберігати анонімність клієнта [26]. Структурна схема роботи клієнт серверного зв’язку показана на рисунку 3.1

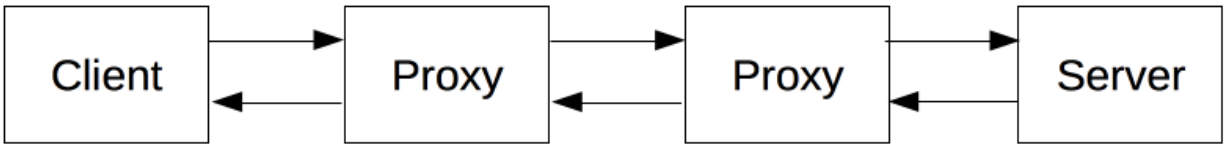


Рисунок 3.1 – Структурна схема роботи клієнт серверного зв’язку

Протокол MQTT – це спрощений мережевий протокол, що працює на TCP/IP. Використовується для обміну повідомленнями між пристроями за принципом видавець-підписник [27]. До переваг MQTT протоколу можна віднести:

– простий у використанні. Протокол є програмним блоком без зайвої функціональності, що може бути легко вбудований в будь-яку складну систему;

– зручний для більшості рішень з датчиками. Дає можливість пристроям виходити на зв'язок і публікувати повідомлення, які не були заздалегідь відомі або визначені;

– легкий у адмініструванні;

– низьке навантаження на канал зв'язку;

– робота в умовах постійної втрати зв'язку або інших проблем на лінії;

– немає обмежень на формат переданого контенту.

MQTT визначає методи (так звані «дієслова»), щоб вказати бажану дію, яка повинна виконуватися на ідентифікованому ресурсі. Чим є цей ресурс, будь то вже існуючі дані або дані, що генеруються динамічно, залежить від реалізації сервера. Часто ресурс відповідає файлу або результату виконання файлу, розміщеного на сервері. MQTT підтримує наступні методи:

– Connect (З'єднати) - чекає установки з'єднання з сервером;

– Disconnect (Роз'єднати) - чекає доки клієнт MQTT закінчить будь-яку роботу, що має зробити, і доки роз'єднається TCP/IP сесія;

– Subscribe (Підписатися) - чекає на завершення методу Subscribe чи UnSubscribe;

– UnSubscribe (Відписатися) - просить сервер відписати клієнта від

одного або кількох тем;

– Publish (Публікувати) - негайно повертається в потік додатку після того, як передає запит клієнту MQTT.

При відправленні повідомлень за допомогою протоколу MQTT можна обирати наступні рівні якості доставки:

– 0 — повідомлення може бути доставлено щонайбільше раз (або не доставлено);

– 1 — повідомлення буде доставлено щонайменше раз (а може й більше).

– 2 — повідомлення буде доставлено рівно один раз [27].

Якщо обирати з цих двох варіантів описаних вище протокол для передачі даних з девайсу до платформи то більшим вигідним є MQTT протокол, адже він є більш “легким” завдяки тому, що передає данні як масиви байт. Це надає змогу витрачати меншу кількість байт для передачі інформації порівняно з протоколом HTTP. Кількість переданих байт є важливим показником, особливо для девайсів які використовують для передачі інформації мобільний інтернет, адже як правило кількість інформації яку можна передати є лімітованою пакетом послуг який надає мобільний оператор.

Ще однією перевагою є те що MQTT протокол працює по моделі видавець-підписник. Це дозволяє скоротити витрати батареї девайсу. Але беручи до уваги той факт, що HTTP наразі є більш популярним протоколом який використається у великій кількості сфер, а платформа повинна надавати якомога ширші можливості для підключення пристроїв можна зробити висновок, що платформа повинна мати підтримку як HTTP так і MQTT протоколів.

3.2.5 Безпека

Інформаційна безпека є дуже важливим фактором, адже дані користувачів та їх девайсів повинні бути захищені і не доступними для інших користувачів. Так, користувач який підключив декілька девайсів до платформи та хоче

отримувати з них інформацію повинен бути впевнений, що цю інформацію може отримати тільки він, якщо він не хоче щоб вона була публічно.

Для вирішення цієї проблеми потрібно заборонити користувачам будь який доступ напряму до бази даних. Основною загрозою при доступі користувачів до бази даних окрім того, що вони зможуть бачити інформацію з датчиків усіх користувачів, є те що вони зможуть змінювати інформацію або навіть видаляти її. Зазвичай, для того щоб база даних не була доступна для кінцевих користувачів вона розміщається в приватній мережі. Це дозволяє бути впевненим в тому, що ніхто з зовні не зможе до неї приєднатися.

Також, обов'язковим є створення користувача який буде мати можливість виконувати лише ті операції на які у нього є права та встановлення для такого користувача паролю. Данні про користувача та пароль повинні зберігатися в сховищі з обмеженим доступом.

Веб сервіс який буде надавати інформацію користувачам повинен мати доступ до сховища де знаходиться інформація про користувача та пароль до бази даних так як для того щоб працювати з даними з бази даних веб сервіс повинен встановити з нею зв'язок. Також веб сервіс повинен мати власні засоби захисту для того щоб ідентифікувати користувачів які виконують запити. Це дозволяє перевіряти чи має користувач доступ до девайсів з яких він хоче отримати інформацію та чи авторизований він в платформі взагалі.

Для побудування такої системи захисту, як правило використовуються сервер авторизації. Він зберігає інформацію про всіх користувачів які зареєструвалися на ресурсі, дозволяє виконувати авторизацію за допомогою логіну та паролю, а також створювати нові аккаунти. Такий сервер зберігає у власній базі даних інформацію про користувачів за допомогою чого він може перевіряти данні які відправляє користувач під час авторизації.

Після того як сервіс успішно авторизував користувача у відповідь він відправляє підписаний токен з інформацією про користувача. Як правило токен передається у форматі JWT.

JWT - це стандарт токена доступу на основі JSON, стандартизованого в

RFC 7519. Використовується для верифікації тверджень[28]. JWT складається з трьох основних частин:

- заголовок;
- вміст;
- підпис.

Заголовок, як правило, складається з двох частин - токену та алгоритму підпису. Вміст складається з вимог які описують інформацію про користувача та додаткову інформацію. Існує три види вимог:

- зарезервовані – це вимоги які є не обов’язковими але імена яких є зарезервованими. Приклади зарезервованих вимог – iss(той хто видав токен) та exp(час закінчення дії токену);
- публічні – це вимоги які є публічними;
- приватні – це вимоги до яких мають доступ тільки довірені ресурси.

Після того як користувач отримує токен він може за допомогою нього, на час поки не вичерпався час дії токену, як правило приблизно 24 години, виконувати запити до серверів які очікують отримати такий токен для аутентифікації та отримувати з них данні.

Є велика кількість вже готових рішень для серверів аутентифікації. Вони поділяються на два типи – сервіси які надають послуги аутентифікації та сервери аутентифікації написані власноруч. Сервіси які надають вже готові рішення для послуг аутентифікації дозволяють не витрачати час на написання власного серверу а також на його захист. Це пришвидшує процес розробки але всі сервіси даного типу є платними. Найбільш відомі з них – це Okta та Oauth0. Написані власноруч сервіси потребують часу на їх імплементацію а також великі зусилля за для того щоб тримати їх у безпеці.

Розглянемо детальніше спосіб авторизації та роботи з сервером за допомогою сервісу Auth0 показаний на рисунку 3.2.

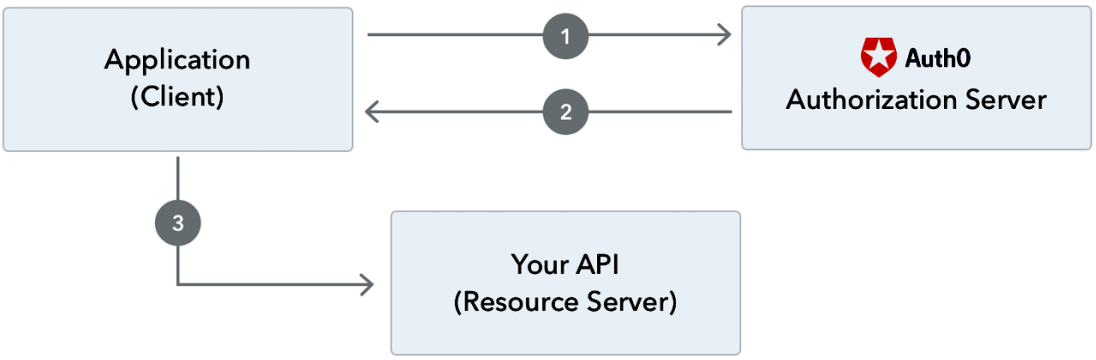


Рис. 3.2 – Авторизація за допомогою Auth0

Клієнт намагається авторизуватися у сервісі Auth0 за допомогою надсилання свого логіну та паролю (шаг 1). Як правило у ролі клієнта виступає веб клієнт або мобільний додаток. У відповідь сервіс Auth0 надсилає JWT токен у разі, якщо авторизація пройшла успішно, або повідомлення що авторизація була не успішною у разі якщо, наприклад, було вказано неправильний логін чи пароль або такого користувача не існує (шаг 2).

У разі отримання токену, клієнт має можливість надсилати запити до серверу для отримання необхідної інформації. Сервер, спочатку, перевірить токен а також час закінчення його дії, та у разі успішної перевірки відправить у відповідь інформацію яка була запитана(шаг 3). Існує велика кількість варіантів проходження процесу отримання токену від серверу аутентифікації. Одним з основних варінтів, який зараз використовується для отримання токену в веб додатку є PKCE(Proof Key for Code Exchange). Спосіб використання цього підходу показаний на рисунку 3.3. PKCE має наступний спосіб роботи:

– користувач натискає кнопку входу в додаток;
– додаток створює випадковий криптографічний верифікатор;
– за допомогою створеного верифікатора додаток створює виклик;
– додаток перенаправляє користувача на сервер авторизації;
– сервер авторизації перенаправляє користувача на сторінку авторизації;
– користувач реєструється за допомогою однієї із запропонованих опцій;

– сервер авторизації зберігає у себе виклик та перенаправляє користувача назад до додатку разом з згенерованим кодом;

– додаток надсилає код та верифікатор до сервера авторизації;

– сервер авторизації надсилає токен;

– додаток відправляє токен до серверу з даними;

– Сервер перевіряє токен та повертає дані.

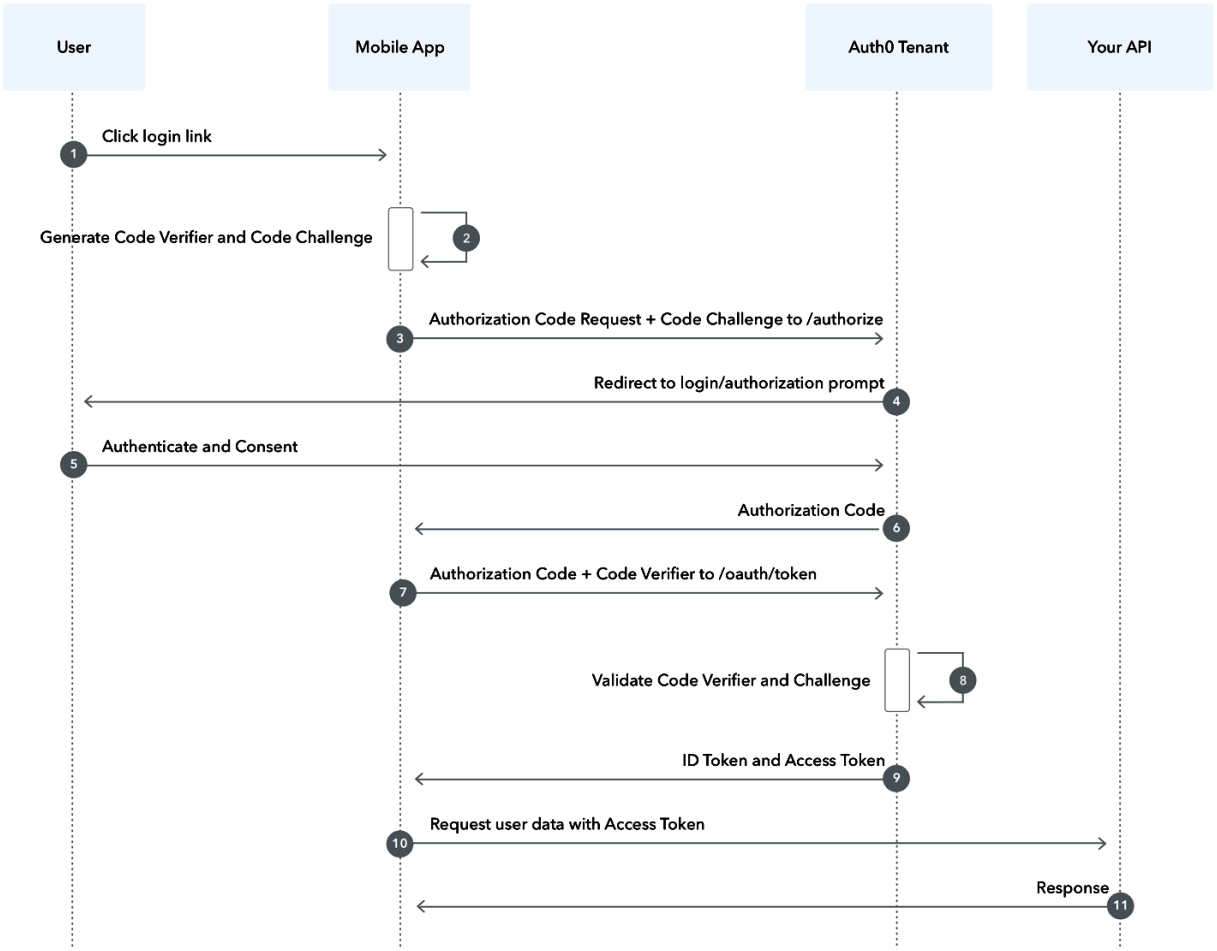


Рис.3.3 – Схема роботи PKCE

Цей метод дозволяє уникнути викрадення токену зловмисниками так як вони можуть отримати лише код авторизації але без верифікатора він не має ваги.

З точки зору девайсів які підключаються до платформи теж існує необхідність в проходженні верифікації. Ця необхідність виникає через те, що

платформа повинна додавати тільки ті девайси які попередньо пройшли верифікацію та були зареєстровані щоб уникнути ситуацію коли до платформи підключаються девайси які розробив, скажімо, користувач власноруч та не зареєстрував. Цього потрібно уникати тому що такий підхід не дозволяє слідкувати за цими девайсами та дозволяє користуватися ресурсами платформи безмежно.

Також, не верифіковані девайси не можна буде пов'язати з аккаунтами користувачів, що призведе до того що користувачі не зможуть отримати з них інформацію. Для вирішення цих проблем необхідно зберігати JWT токен на кожному девайсі. Також можна використовувати технологію RSA. RSA – це криптографічний алгоритм з відкритим ключем, що базується на обчислювальній складності задачі факторизації великих цілих чисел. Алгоритм RSA складається з 4 етапів: генерації ключів, шифрування, розшифрування та розповсюдження ключів. Безпека алгоритму RSA побудована на принципі складності факторизації цілих чисел. Алгоритм використовує два ключі — відкритий (public) і секретний (private), разом відкритий і відповідний йому секретний ключі утворюють пари ключів (keypair).

Відкритий ключ не потрібно зберігати в таємниці, він використовується для шифрування даних. Якщо повідомлення було зашифровано відкритим ключем, то розшифрувати його можна тільки відповідним секретним ключем [31]. Цей механізм дозволяє зберігати на девайсі відкритий ключ, а на сервері закритий, та при надсиланні даних з девайсу перевіряти відкритий ключ який надходить разом з даними і тільки у разі успішної перевірки обробляти отримані данні.

3.3 Розробка структурної схеми

Беручі до уваги проблеми та способи їх вирішення описані у попередньому розділі а також вимоги до платформи структурна схема платформи має вигляд показаний на рисунку 3.4.

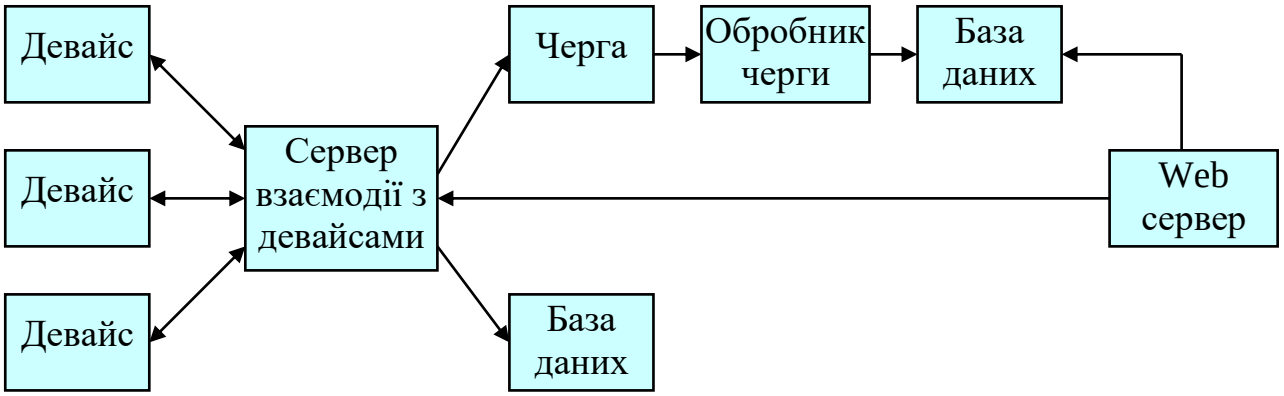


Рисунок 3.4 – Структурна схема

Сервер взаємодії з девайсами забезпечує комунікацію між девайсами та платформою. Він дозволяє приймати повідомлення з девайсів за допомогою протоколів HTTP та MQTT. Також він має власну базу даних яка зберігає інформацію про девайси які булу зареєстровано, що дозволяє перевіряти девайси, що підключаються до серверу на те що вони були зареєстровані системою. Також сервер перевіряє публічний ключ який надходить разом з повідомленнями з девайсів у цілях безпеки. Якщо девайс з якого відправили повідомлення не був знайдений в базі даних серверу взаємодії з девайсами або повідомлення має невірний публічний ключ то таке повідомлення буде проігноровано. У разі вдалої обробки повідомлення воно буде записано у чергу.

Черга зберігає у собі повідомлення які надійшли з сервера взаємодії з девайсами до того моменту поки обробник черги їх не обробить. Цей механізм дозволяє зберігати значно більшу кількість повідомлень ніж обробник має можливість обробити в один момент в часі, що дозволяє не втрачати повідомлення які надходять з девайсів.

Обробник черги послідовно вичитує повідомлення з черги та записує їх в базу повідомлень. База повідомлень зберігає в собі інформацію про повідомлення отримані з девайсів. Черга, обробник черги та база повідомлень утворюють потоковий ETL процес. Веб сервер дозволяє користувач отримувати інформацію про повідомлення які надійшли з девайсів які він вичитує з бази даних для повідомлень. Також він дозволяє відправляти команди до девайсів

так як він може надсилати запити до серверу взаємодії з девайсами який в свою чергу відправляє повідомлення на сам девайс. Також за допомогою веб серверу користувачі мають змогу управляти своїми девайсами а також додавати нові.

3.4 Висновки

Структурна схема розроблена у даному розділі дозволить вирішити основні проблеми, а саме обробку великої кількості даних, безпеку збереження даних, а також надання єдиного інтерфейсу користувачу для перегляду даних які надходять з девайсів власником яких він є. Застосування черги, обробника черги та бази даних дозволяє застосувати механізм ETL, що дозволить обробляти велику кількість даних.

4 ПОБУДОВА ПЛАТФОРМИ

4.1 Google Cloud Platform

Для реалізації платформи було вирішено використовувати хмарне рішення від Google - Google Cloud Platform. Google Cloud Platform (або GCP) – це запропонований компанією Google набір хмарних служб, які виконуються на тій же самій інфраструктурі, яку Google використовує для своїх продуктів призначених для кінцевих споживачів, таких як Google Search та YouTube. Окрім інструментів для керування, також надається ряд модульних хмарних служб, таких як обчислення, зберігання даних, аналіз даних та машинне навчання [32].

GCP на сьогодні є дуже популярним рішенням яке використовує велика кількість компаній для побудування за допомогою нього своїх власних продуктів. Однією з причин такої популярності є те, що всі сервіси користувачів запускаються в дата-центрах компанії Google. Це дозволяє не витрачати час на побудування свого власного дата-центру, а в подальшому не слідкувати за ними, та не оновлювати їх комплектуючі які з часом будуть застарівати та потребувати заміни.

Ще однією перевагою використання GCP є надання компанією угоди про рівень послуг. Цей рівень описує скільки часу в рік сервіси клієнту будуть працювати без перебоїв в хмарному рішенні від Google. Як правило, ці показники є набагато кращими аніж може забезпечити свій власний дата-цент адже як уже було сказано раніше, всі сервіси клієнтів працюють на тій самій інфраструктурі як і сервіси Google у яких дуже високі стандарти та вимоги роботи без перебоїв.

Також, окрім високих гарантій та надання власних серверів для запуску на них сервісів клієнтів, GCP надає велику кількість власних сервісів які набагато спрощують розробку власних продуктів.

Хмарне рішення від Google надає дуже велику кількість власних сервісів,

розглянемо основні з них:

– Compute Engine – віртуальні машини, які працюють в дата-центрах Google. Дозволяють виконувати майже будь які обчислення та запускати власні сервіси. Віртуальні машини підтримують масштабування, що означає що у разі зростання навантаження на сервіс який працює всередині віртуальної машини, кількість віртуальних машин буде збільшена, що дозволить обробляти більше даних, а у разі якщо навантаження буде зменшено то кількість віртуальних машин теж може бути зменшена. Це дозволяє оптимізувати витрати на віртуальні машини;

– Cloud Storage – це сховище об’єктів. Дозволяє зберігати об’єкти в дата-центрах Google та контролювати рівень доступу. Зазвичай використовується для зберігання файлів різного типу, в тому числі відео, фото та музичних файлів;

– Cloud Pub/Sub – це повністю автоматизована система доставки повідомлень в реальному часі, яка дозволяє пересилати повідомлення між незалежними додатками;

– BigQuery – це повністю автоматизована та висок масштабована реляційна база даних яка предназначена для збереження та обробки великих даних. Вона допомагає зробити аналіз даних більш продуктивним за допомогою вистроєного движку обробки даних;

– App Engine – це повністю автоматизований сервіс для побудови власних веб серверів який дозволяє не витрачати час на побудування інфраструктури та налаштування програмного забезпечення. App Engine підтримує велику кількість мов програмуваня в тому числі – Java, Go, Node.js та Python.

4.2 Розробка архітектури платформи на базі GCP

Беручі до уваги вимоги платформи а також структурну схему розроблену

у попередньому розділі, структурна схема на базі GCP показана на рисунку 4.1.

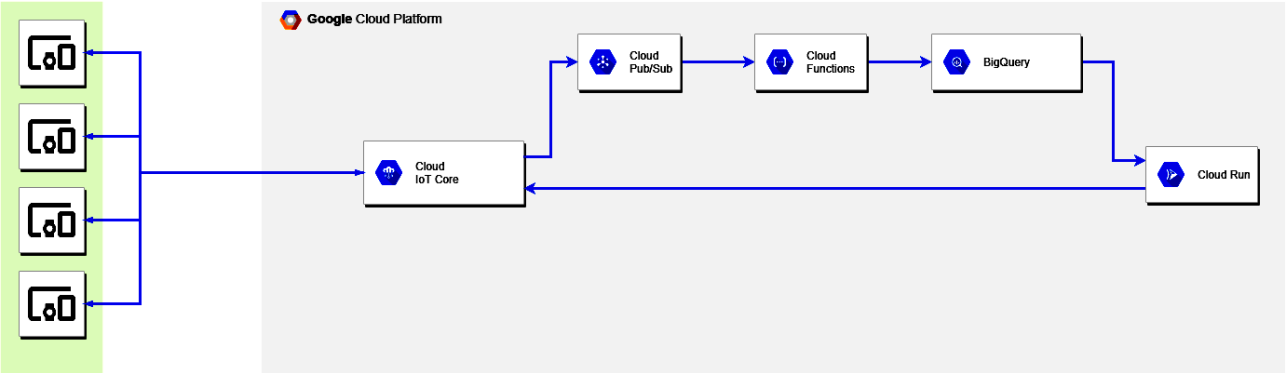


Рисунок 4.1 – Структурна схема платформи на базі GCP

У реалізації використовується декілька сервісів GCP, а саме Cloud IoT Core, Pub/Sub, Cloud Functions, BigQuery та Cloud Run. Розберу можливості та роль кожного з них.

4.2.1 Cloud IoT Core

Cloud IoT Core – це повністю керований сервіс який використовується для підключення девайсів та отримання з них даних. Сервіс має свою власну базу даних де зберігається інформація про девайси які зареєстровані в сервісі. Це дозволяє ідентифікувати девайс який надсилає повідомлення. Також сервіс надає механізм аутентифікації девайсів. Це досягається за допомогою зберігання приватного RSA ключа для кожного з девайсів.

В свою чергу кожен девайс повинен мати публічний RSA ключ та надсилати його під час запиту. У разі успішної перевірки, повідомлення яке було надіслано з девайсу буде оброблене. У разі якщо перевірка не була пройденою повідомлення буде відхиленим.

В платформі Cloud IoT Core виступає єдиною точкою входу для повідомлень з девайсів, а також сервісом як дозволяє відправляти команди до девайсів. Сервіс підтримує як HTTP так і MQTT. Сервіс є повністю керованим та само-масштабованим. Це означає, що немає необхідності в налаштовані

сервісу в залежності від навантаження. Натомість, сервіс здатен масштабуватися сам. Після отримання повідомлень з девайсу та успішної аутентифікації, повідомлення вправляється в Pub/Sub, що дозволяє знизити навантаження на обробника повідомлень. Під час створення аккаунту та реєстру в Cloud IoT Core також створючаться і Pub/Sub топік в який будуть надходити данні з девайсів.

4.2.2 Cloud Pub/Sub

Cloud Pub/Sub – це повністю керований сервіс доставки повідомлень у реальному часі який дозволяє відправляти та отримувати повідомлення. Сервіс працює по моделі публікація-підписка.

Публікація-підписка – це шаблон проектування в архітектурі програмного забезпечення, шаблон поведінки, що реалізує механізм передачі повідомлень, в якому відправники повідомлень, які називаються видавцями (publishers), не здійснюють пряме відправлення повідомлень приймачам, які називаються підписниками (subscribers), замість цього опубліковані повідомлення розбиваються на категорії за класами, без знання про те, яким підписникам вони мають бути прийняті і чи взагалі будуть такі підписники.

Аналогічно, підписники виявляють зацікавленість в певних класах повідомлень і приймають ті повідомлення, які їх цікавлять, без знання того, які видавці їх публікують. Шаблон сприяє отриманню програмного забезпечення з більшою масштабованістю та динамічнішою топологію мережі [34]. Основними компонентами сервісу є:

- топік – ресурс в який видавець надсилає повідомлення;
- підписка – ресурс який зберігає в собі повідомлення з топіку для додатків які на нього підписані;
- повідомлення – Набір даних які надсилає видавець до топіку;
- атрибути повідомлення – пари ключ/значення за допомогою яких можна вказувати додаткові атрибути для повідомлення.

Приклад інтеграцій за допомогою сервісу Cloud Pub/Sub в середині платформи показані на рисунку 4.2. Як приклад, Cloud Pub/Sub здатен отримувати повідомлення про зміни у документах в сервісі збереження об’єктів Cloud Storage або повідомлення з сервісу віртуальних машин Compute Engine та надсилати їх до сервісу веб додатків App Engine.

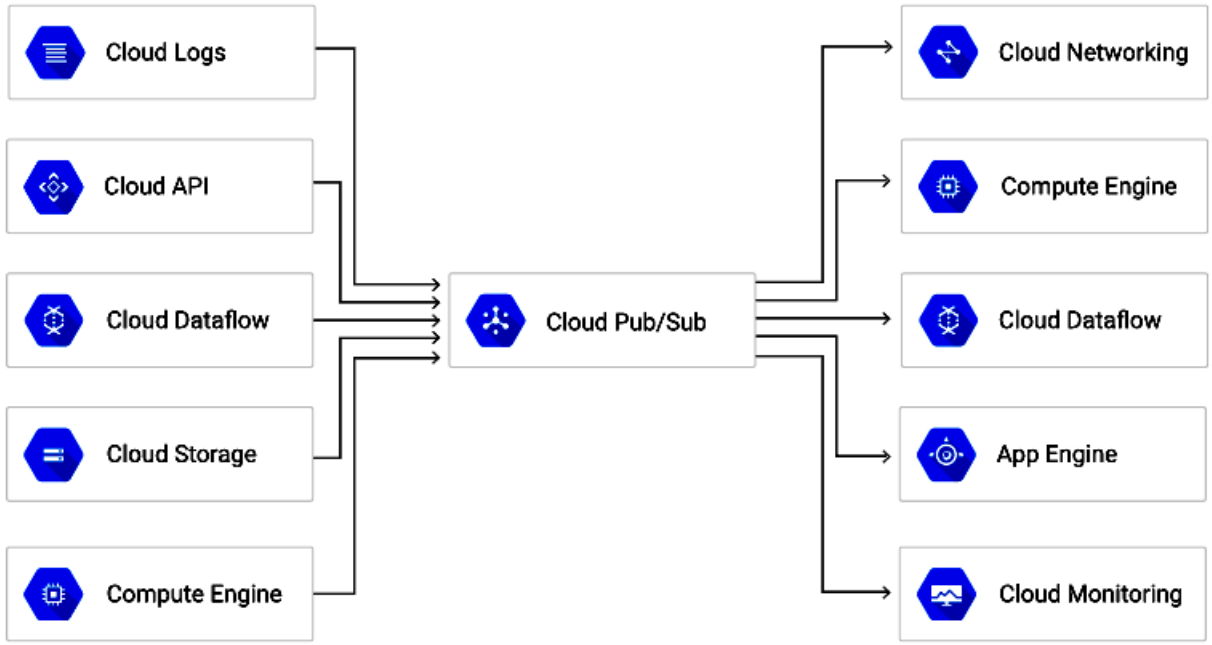


Рисунок 4.2 – Приклад інтеграцій за допомогою Cloud Pub/Sub

Окрім внутрішньої взаємодії всередині платформи, Cloud Pub/Sub також дозволяє отримувати повідомлення з зовнішніх систем. Приклад можливих варіантів зовнішніх систем та їх інтеграція з Pub/Sub показаний на рисунку 4.3.

Так, наприклад, підтримується прямий доступ до топіку з зовнішніх систем за допомогою відключення їх до мережі Google Cloud Platform. Окрім зовнішніх доступ до топіку також можуть отримати веб та мобільні додатки а також IoT девайси.

Це досягається за допомогою додаткового серверу який знаходиться поміж девайсами чи додатками та топіком. Він дозволяє отримувати повідомлення по протоколу HTTP чи MQTT та надсилати їх до топіку. Саме такий підхід використається в сервісі Cloud IoT Core для публікації повідомлень з девайсів до топіку.

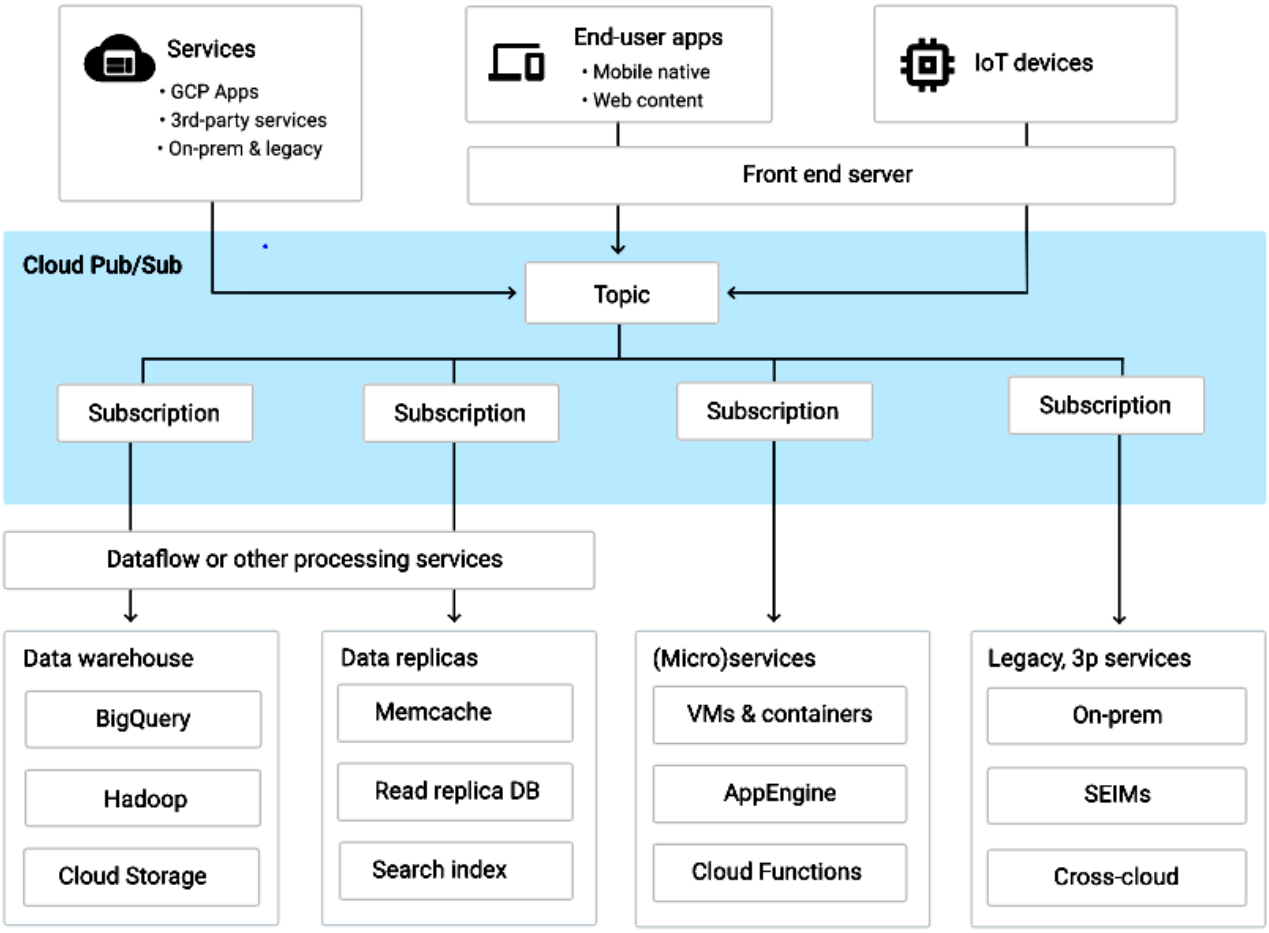


Рис. 4.3 – Приклад взаємодії з Pub/Sub з зовнішніх ресурсів

4.2.3 Cloud Functions

Cloud Functions – це сервіс який надає безсерверні обчислення. Безсерверні обчислення – це модель хмарних обчислень для яких платформа динамічно керує виділенням машинних ресурсів. Іноді безсерверні обчислення також іменують «Функція як послуга» (англ. Function as a Service, FaaS), тому що одиницею коду є функція, яка виконується платформою. По суті для виконання одного запиту створюється окремий контейнер, який знищується після виконання [35].

Безсерверні обчислення сьогодні мають дуже велику популярність серед розробників програмного забезпечення. Основною причиною їх популярності є простота налаштування в порівнянні зі звичайними серверами чи віртуальним машинами. Адже, користуючись, наприклад, віртуальною машиною, для того

щоб налаштувати сервер, окрім написання самого коду який буде використаний необхідно встановити всі необхідні пакети для його запуску, а також підтримувати його у разі якщо після запуску виникнуть проблеми. Як правило це забирає велику кількість часу а також додає відповідальності розробникам програмного забезпечення.

В порівняння, використовуючи безсерверних обчислення, немає необхідності в налаштуванні для нього інфраструктури і потребує лише написання коду. Звичайно, теології безсерверного обчислення теж використовують сервери для запуску на них коду, але на відміну від звичайних серверів деталі реалізації та налаштування серверів скриті від розробника і він не має до них доступу.

Це дозволяє сконцентруватися лише на розробці коду та не витрачати час на налаштування серверів. Звичайно технологія має свої недоліки. Так як сервери для безсерверних обчислень налаштовуються хмарним провайдерами вони підтримають лише деякі мови програмування для написання коду який буде виконуватися. В сервісі Cloud Functions підтримуються мови програмування Node.js та Go.

Ще одним недоліком є можливе затримання під час початку виконання функції. Це зумовлено тим фактом, що кожна функція виконується у власному середовищі яке потребує часу на створення під час запуску функції.

Одним з архітектурних стилів побудування програмних додатків який використається в розробці програмних додатків за допомогою безсерверних обчислень є мікросерісна архітектура. Мікросервісна архітектура – це підхід, при якому єдиний додаток будується як набір невеликих сервісів, кожен з яких працює у власному процесі і спілкується з іншими використовуючи легкі механізми, як правило HTTP.

Ці сервіси побудовані навколо бізнес-потреб і розгортаються незалежно, з використанням повністю автоматизованого середовища. Існує абсолютний мінімум централізованого управління цими сервісами. Самі по собі ці сервіси можуть бути написані на різних мовах і використовувати різні технології

зберігання даних[36]. Основними принципами побудування системи під час побудування рішення за допомогою мікросервісів є:

– маштабованість – це можливість сервісу збільшувати свої ресурси у разі якщо навантаження на нього збільшилася та зменшувати у разі якщо наразі сервіс має більше ресурсів аніж потрібно для обробки поточного навантаження. Основна проблема яка виникає під час побудування масштабованих сервісів це розподілення стану сервісу між усіма його екземплярами. Це необхідно тому що сервіс ніколи не знає на який саме його екземпляр буде надіслано наступний запит і якщо, наприклад, сервіс зберігає інформацію яка необхідна конкретному користувачу на одному екземплярі, то існує ризик що під час запиту на користувача йому відповість інший екземпляр який не має даних необхідних користувачу, що призведе до то що користувач не отримає данні які він запросив. Як правило ця проблема вирішується за допомогою винесення стану за рамки сервісу, наприклад, в базу даних чи кеш. Так як база чи кеш є єдиними для всіх сервіс то стає не важливо на який саме екземпляр сервісу відправить запит користувач – він завжди отримає данні які він запитав;

– доступність – мікросервіси повинні бути постійно доступними. Це пов’язано з тим, що в мікросервісній архітектурі сервіси надсилають запити до інших сервісів які в свою чергу теж надсилають запити до інших сервісів. Це призводить до ситуації коли у разі недоступності одного з сервісів в ланцюгу викликів, сервіси один за одним перестануть працювати адже не зможуть отримати данні з інших сервісів які їм необхідні для обробки запиту. Ця проблема вирішується комплексно. З одного боку кожен сервіс повинен мати механізми перешкоджання ситуацій коли всі екземпляри сервісу є недоступними. Як правило це забезпечується розділенням екземплярів по різним дата-центра, тож у випадку якщо один з дата-центрів буде тим часово недоступний, то екземпляри які знаходяться у іншому дата-центрі зможуть продовжити приймати запити. З іншого боку кожен сервіс який викликає інші сервіси повинен мати механізми які дозволять йому обробляти ситуацію коли сервіси з яких він отримує данні не доступні. Це дозволить не обривати весь

ланцюг викликів у разі якщо один з сервісів не доступний, а натомість обробити цю ситуацію у сервісі який викликає недоступний сервіс та продовжити виконання ланцюга запитів;

– незалежність – кожен мікросервіс повинен бути незалежним від інших. Це дозволяє уникнути ситуацій коли при зміні кодової бази іншого мікросервісу з яким сервіс має спільну кодову базу сервіс перестає працювати корректно через то, що зміни які були зроблені не співпадають з очікуваннями сервісу. Ця проблема вирішується за допомогою повного розділення кодової бази інфраструктури поміж різними мікросервісами. Ця практика також називається “нещільне з’єднання”. Використовуючи такий підхід мікросервіси спілкуються між собою виключно за допомогою синхронних чи асинхронних протоколів, таких як HTTP, AMQP та інших.

В мікросервісній архітектурі одним з головних принципів є подрібнення основної бізнес моделі на більш малі компоненти які будуються у виді мікросервісів. Це дозволяє кожному з сервісів сконцентруватися лише на одній бізнес задачі. Це дозволяє спростити розуміння системи в цілому адже в порівнянні якщо збудувати один сервісів який буде містити в собі всю бізнес модель то з часом його стане складно підтримувати адже його складність буде постійно зростати.

Хмарні функції дуже легко використовувати під час побудування мікросервісної архітектури, адже кожна функція є відокремленим сервісом який виконує тільки одну бізнес функцію. Так як зобов’язання про доступність хмарних функцій бере на себе їх постачальник, це дозволяє добитися великих показників доступності. Також хмарні функції чудово масштабуються, адже кожна з них під час запиту розгортається в одній з віртуальних машин в дата-центрі. Це дозволяє обробляти велику кількість даних одночасно.

Ще одним способом використання хмарних функцій є побудування сервісів які повинні виконуватися під час створення події. Наприклад, зміни стану бази даних чи надходженні нової інформації з девайсу. Сервіс Cloud Functions може обробляти події створені з різних джерел. Процес надходження

подій показаний на рисунку 4.4.

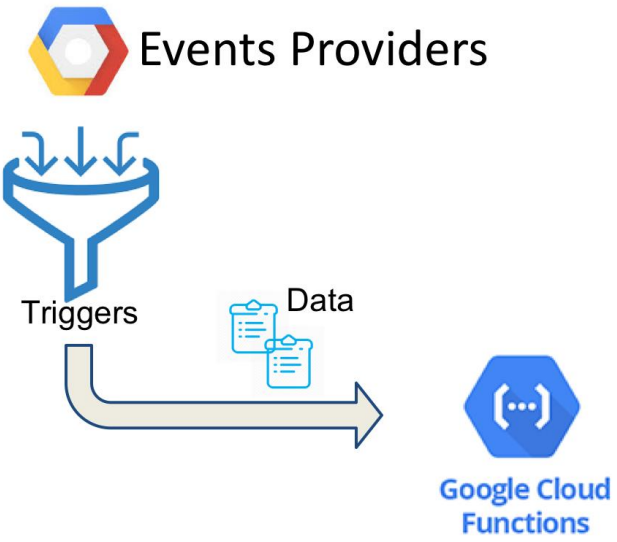


Рисунок 4.4 – Надходження подій до сервісу Google Cloud Functions

Сервіс підтримує різні джерела з яких надходять події, а саме події які відбуваються:

– під час додавання повідомлення в сервіс Cloud Pub/Sub;

– під час додавання, оновлення чи видалення об’єкту в сервісі Cloud Storage;

– під час запити по протоколу HTTP;

– під час зміни даних у сервісі Stackdriver Logging який дозволяє зберігати та файли реєстрації;

– під час створення події за допомогою сервісу Firebase який, як правило, використовують для побудування мобільних додатків.

Сервіс Cloud Functions має велику сферу можливих варіантів використання. Деякими з них є:

– побудова серверу для мобільних додатків;

– побудова системи обробки подій у реальному часі;

– побудова системи обробки інформації з IoT девайсів;

– побудова системи аналізу зображень та відео.

В платформі сервіс Cloud Functions використовується для обробки даних які надсилаються з девайсів. Основу задачу яку вирішує сервіс є швидка

обробка та збереження даних в сервіс Big Query. Так як задача є простою, немає необхідності в побудування складного рішення на базі одного з сервісів які надають можливість запускати власні у сервери. Сервіс Cloud Functions здатний витримувати велике навантаження, отже, велика кількість повідомлень з девайсів які поступають на обробку через сервіс Cloud Pub/Sub не буде проблемою. Код функції яка обробляє повідомлення з черги та записує в BigQuery наведений в додатку А.

4.2.4 BigQuery

BigQuery – це безсерверне сховище даних, для інтерактивного широкомасштабного аналізу великих наборів даних. Може використовуватись через веб інтерфейс, інтерфейс командного рядка та API[37]. Сервіс BigQuery створений в першу чергу для обробки та аналітики великих даних. Основними особливостями цього сервісу є:

– безсерверність – немає потреби в створенні та підтримці ресурсів які необхідні для роботи сервісу адже всю роботу по оновленням, безпеці та управлінні інфраструктурою виконує хмарна платформа, що дозволяє сконцентруватися на роботі з даними а не на підтримці стійкості сервісу;

– аналітика у реальному часі – велика швидкість додавання нових даних дозволяє у реальному часі отримувати нові данні для аналізу;

– висока доступність – сервіс забезпечує високу доступність сервісу за допомогою зберігання даних у кількох місцях одночасно;

– стандарт SQL – сервіс надає підтримку стандарту SQL, що дозволяє не витрачати час на вивчення нового синтаксису;

– обробка зовнішніх джерел – сервіс здатний обробляти та об’єднувати данні з зовнішніх ресурсів, таких як Cloud Storage, Cloud Bigtable та електронні таблиці;

– розділення сховища та механізму обчислень – сервіс має відокремлені ресурси для зберігання та обробки даних. Це дозволяє обирати сховище та

варіанти обробки які підходять для поставленої задачі;

– резервне копіювання – сервіс автоматично копіює дані та зберігає історію змін на протязі семи днів. Сервіс також дозволяє легко відновити данні з різних часових проміжків;

– геопросторові типи даних – сервіс підтримує геопросторові типи даних. Геопросторові типи даних - це інформація, що визначає географічне положення та характеристики природних та побудованих об'єктів та кордони на поверхні Землі. Дані про об'єкти та явища, які безпосередньо або опосередковано пов'язані з місцеположенням на Землі, що визначені у певній системі просторово-часових координат; набори даних про такі об'єкти та зв'язки між ними. Базовий набір геопросторових даних — стандартизована сукупність загальногеографічних даних, покладених в основу інтегрування і спільного використання у геоінформаційних системах геопросторових даних різноманітного походження[38]. BigQuery забезпечує підтримку довільних точок, ліній та багатокутників в форматах WKT та GeoJSON. Розглянемо кожен з цих форматів. WKT – це текстовий формат представлення векторної геометрії та опису системи координат. Цей формат, як правило, використовується для надання спільного інтерфейсу для обміну просторовими даними між різними програмами та сервісами .В свою чергу GeoJSON – це відкритий формат призначений для зберігання географічних структур даних, заснований на JSON. Формат може зберігати примітивні типи для опису географічних об'єктів, такі як: точки (адреси та місця розташування), лінії (вулиці, шосе, кордони), полігони (країни, штати, ділянки землі). Також можуть зберігатися так звані мультитипи, які представляють собою об'єднання декількох примітивних типів. Формат GeoJSON відрізняється від інших стандартів ГІС тим, що він був написаний і підтримується не організацією зі стандартизації, а за допомогою робочої групи розробників. Подальшим розвитком GeoJSON є ТороJSON, розширення GeoJSON, яке кодує геопросторову топологію, і, як правило, забезпечує менший розмір файлів[39]. Підтримка цих двох форматів сервісом BigQuery дозволяє виконувати геопросторовий аналіз;

– служба передачі даних – сервіс дозволяє автоматично передавати данні з інших сервісів, таких як Google Ads, YouTube а також інших партнерських програм за повністю керованним принципом. Також підтримується передача даних з таких сервісів як Teradata та Amazon S3;

– підтримка обробки петабайтів даних – сервіс дозволяє обробляти петабайти даних з великою продуктивністю;

– безпека даних – сервіс надає можливість забезпечення безпечного доступу до даних за допомогою інтеграції з сервісом Cloud Identity. Окрім цього, всі данні які зберігаються в сервісі є зашифрованими;

– географічний контроль – сервіс надає можливість контролю географічних даних в різних локаціях;

– гнучкі можливості завантаження даних – сервіс дозволяє завантажувати даних з сервісу Cloud Storage або за допомогою передачі тисячами рядків в секунду. Це дозволяє роботи аналіз даних у режимі реального часу;

– підтримка великої кількості мов програмування – сервіс надає клієнтські бібліотеки для Java, Python, Node.js, C#, Go, Ruby та PHP.

Розглянемо внутрішню структуру сервісу (рисунок 4.5). BigQuery працює у декількох центрах обробки даних, кожен з яких має сотні тисяч ядер, десятки петабайт ємності для збереження даних та терабайти пропускної здатності мережі. Великий розмір ресурсів які надає сервіс дозволяє користувачам велику ефективність запитів. В середині, сервіс BigQuery використовує двигун виконання Dremel.

Dremel - це масштабована, інтерактивна спеціальна система запитів для аналізу введених даних лише для читання. Поєднуючи багаторівневе виконання дерева та стовпчастий макет даних, він може виконувати запити агрегації понад трильйонні таблиці рядків за секунди.

Обчислення на основі MapReduce [48]. Dremel оркеструє запит який надходить від користувача, розбиваючи його на частини та повторно збирає

після того як отримує результати з усіх частин розбитого запиту перед відправленням результатів запиту користувачу. Dremel перетворює SQL-запит у дерево виконання. Листя дерева, яке воно називає "прорізами", і важко піднімає читання даних від сервісу Colossus і виконує будь-які необхідні обчислення.

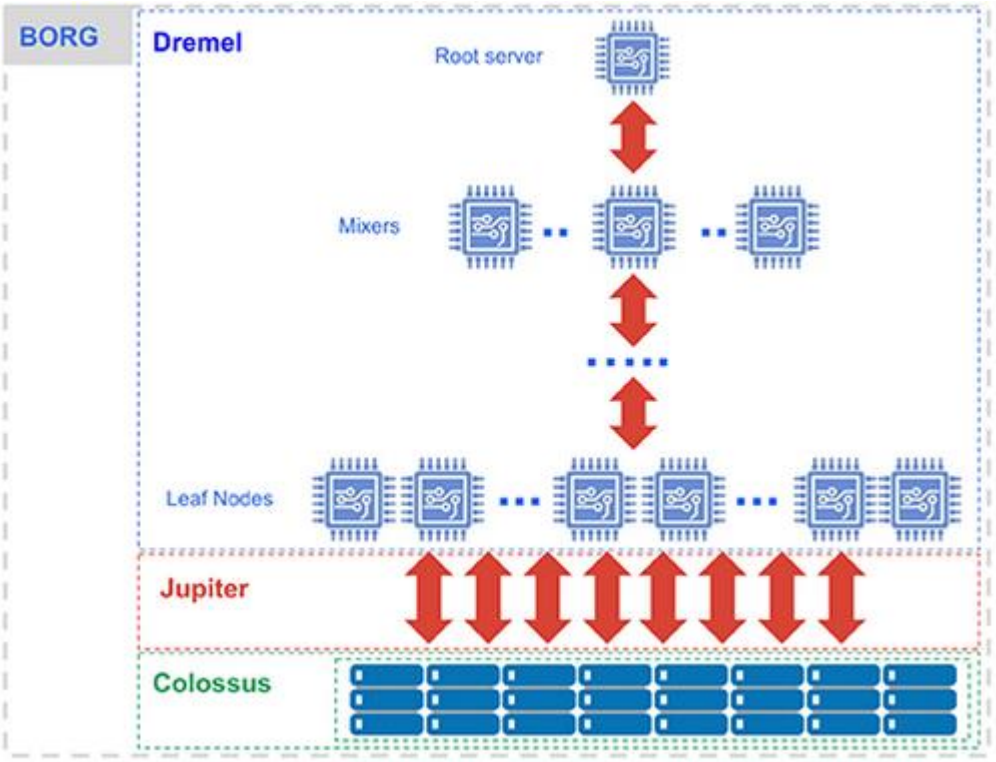


Рисунок 4.5 – Внутрішня реалізація сервісу BigQuery

Гілки дерева - це «змішувачі», які виконують агрегацію. Поміж ними є "перетасовка", яка використовує перевагу мережі "Юпітер" від Google, щоб надзвичайно швидко переміщувати дані з одного місця в інше. Усіма змішувачами керує Borg, який виділяє апаратні ресурси. Dremel динамічно розподіляє слоти для запитів за потребою, підтримуючи справедливість серед декількох користувачів, які надсилають запити одночасно. Один користувач може отримати тисячі слотів для запуску своїх запитів.

BigQuery покладається на Colossus, розповсюджену файлову систему останнього покоління Google. У кожному центрі обробки даних Google є власний кластер Colossus, і кожен кластер Colossus має достатньо дисків, щоб

дати можливість кожному користувачеві BigQuery тисячі виділених дисків одночасно. Colossus також обробляє реплікацію, відновлення (при збою дисків) та розподілене управління (тому немає жодної точки відмови). BigQuery використовує стовпчастий формат зберігання ColumnIO та алгоритм стиснення, щоб зберігати дані в Colossus найбільш оптимальним способом для читання великої кількості структурованих даних.

Сервіс надає можливість інтеграція з екосистемою для обробки великих даних. Завдяки інтеграції з такими сервісам як Cloud Dataproc та Cloud Dataflow, сервіс BigQuery забезпечує інтеграції з екосистемою Apache Big Data, дозволяючи існуючим навантаженням Hadoop/Spark та Beam читати та записувати дані безпосередньо з BigQuery. Це відкриває великі можливості для роботи з великими даними. Це означає, що зберігаючи данні у сервісі BigQuery у випадку необхідності додаткової обробки перед додаванням даних в сховище чи обробки вже збережених даних буде дуже просто додати сторонні обробники які є стандартами у обороті великих даних. Це дозволить додати до платформи можливості аналізу даних які надходять з девайсів для користувачів.

Створемо новий Dataset з назвою telemetry. В dataset створемо таблицю у наступному форматі:

```
[
    {
        “description”: “Device id”
        “name”: “deviceId”
        “type”: “STRING”
        “mode”: “NULLABLE”
    },
    {
        “description”: “Time when metric was saved”
        “name”: “timestamp”
        “type”: “STRING”
        “mode”: “NULLABLE”
```

```
    },
    {
        “description”: “Metric data in JSON format”
        “name”: “data”
        “type”: “STRING”
        “mode”: “NULLABLE”
    }
]
```

Таблиця описує структуру в якій будуть зберігаться данні в середині BigQuery. В таблиці присутні наступні поля:

- deviceId – id девайсу з якого надійшла метрика;
- timestamp – час коли метрика надійшла;
- data – данні які надійшли з девайсу у форматі JSON.

4.2.5 Cloud Run

Для побудування веб-серверу який повинен стати точкою доступу для користувачів в систему та дозволятиме отримувати інформацію про повідомлення які надійшли з девайсів, посилати команди до девайсів, додавати нові девайси та управляти своїм аккаунтом був обраний сервіс Cloud Run. Google Cloud Platform надає різні сервіси які дають можливість на їх базі побудувати необхідне рішення. Розглянемо кожен сервіс детальніше:

4.2.5.1 Compute Engine

Compute Engine – це сервіс який надає високо масштабовані віртуальні машини які працюють у центрах обробки даних Google. Сервіс надає можливість конфігурувати віртуальні машини з бажаним ресурсами а також налаштовувати автоматичне горизонтальне та вертикальне масштабування.

Беручи до уваги той факт, що віртуальні машини знаходяться в центрах обробки Google між якими налаштована велика пропускна спосібність, сервіс Compute Engine дозволяють будувати кластери з віртуальних машин які будуть

мати велику пропускну спосібність для спілкування між віртуальними машинами, що дозолиться знизити час очікування відповіді на запит для користувача.

Завдяки тому, що сервіс надає доступ до віртуальної машини на базі операційної систем Linux або Windows, це надає можливість налаштовувати необхідне програмне забезпечення та не обмежуватися використанням лише тих мов програмування та виконуючого середовища якими обмежені інші сервіси.

Сервіс також надає можливість виконувати пакетну обробку даних. Сервіс надає як HDD сховище розміром до 64 терабайтів так і SSD диски 3 терабайтів. Сервіс має і недоліки. Одним з основних недоліків є те, що так як віртуальна машина яка надається є не налаштованою. Це призводить до необхідності налаштування віртуальної машини перед тим як почати її використовувати.

Хоча, існує програмне забезпечення таке як, наприклад, Ansible, Shef та Puppet яке дозволяє описувати бажаний стан віртуальної машини з всіма необхідними програмним забезпечення та потім автоматично використовувати його для налаштування віртуальних машин, цей процес все одно пов’язаний зі складнощами через те, що конфігурування за допомогою такого програмного забезпечення є досить складним та потребує певних навичок, а також не виключає можливості помилки при налаштуванні.

Переваги:

- можливість гнучкого налаштування необхідних ресурсів;
- підтримка будь яких мов програмування та програмного забезпечення;
- велика пропускна здатність.

Недоліки:

- необхідність налаштування середовища;
- необхідність постійної підтримки та обслуговування ресурсів

4.2.5.2. Kubernetes Engine

Kubernetes Engine – це повністю кероване середовище для запуску контейнерезованих додатків. Контейнер – це стандартна одиниця програмного забезпечення, яка пакує код та всі його залежності, завдяки чому програма швидко та надійно працює в будь якому обчислювальному середовищі[50]. Контейнер використовуються для запуски в них різних рішень, як наприклад к мікросерсівісів чи баз даних.

Контейнери це оптимізований спосіб побудування, тестування, розгортання та повторного розгортання додатку на різних середовищах від ноутбуку до хмарних сервісів. До переваг контейнерів можна віднести:

– менше накладних витрат - Контейнери вимагають менше системних ресурсів, ніж традиційні або апаратні віртуальні машини, оскільки вони не містять зображень операційної системи;

– підвищена портативність - програми, що працюють в контейнерах, можуть легко розгортатися в декілька різних операційних систем та апаратних платформ;

– більш послідовна робота - програми в контейнерах будуть працювати однаково, незалежно від того, де вони розміщені;

– більша ефективність - Контейнери дозволяють додаткам швидше розгортати, виправляти або масштабувати.

Стандартом серед реалізацій технологій контейниризації наразі є Docker. Docker - це операційна система для контейнерів. Подібно до того як віртуальна машина створює віртуальне уявлення апаратного забезпечення сервера (тобто усуває необхідність безпосередньо управляти таким), контейнери створюють віртуальне уявлення серверної операційної системи.

Після установки на кожен сервер Docker надає доступ до простих команд, необхідним для збірки, запуску або зупинки контейнерів [51]. Порівняємо між собою віртуальні машини та контейнери. Віртуальна машина (рисунок 4.6) - це абстракція фізичного обладнання, що перетворює один сервер на багато

серверів.

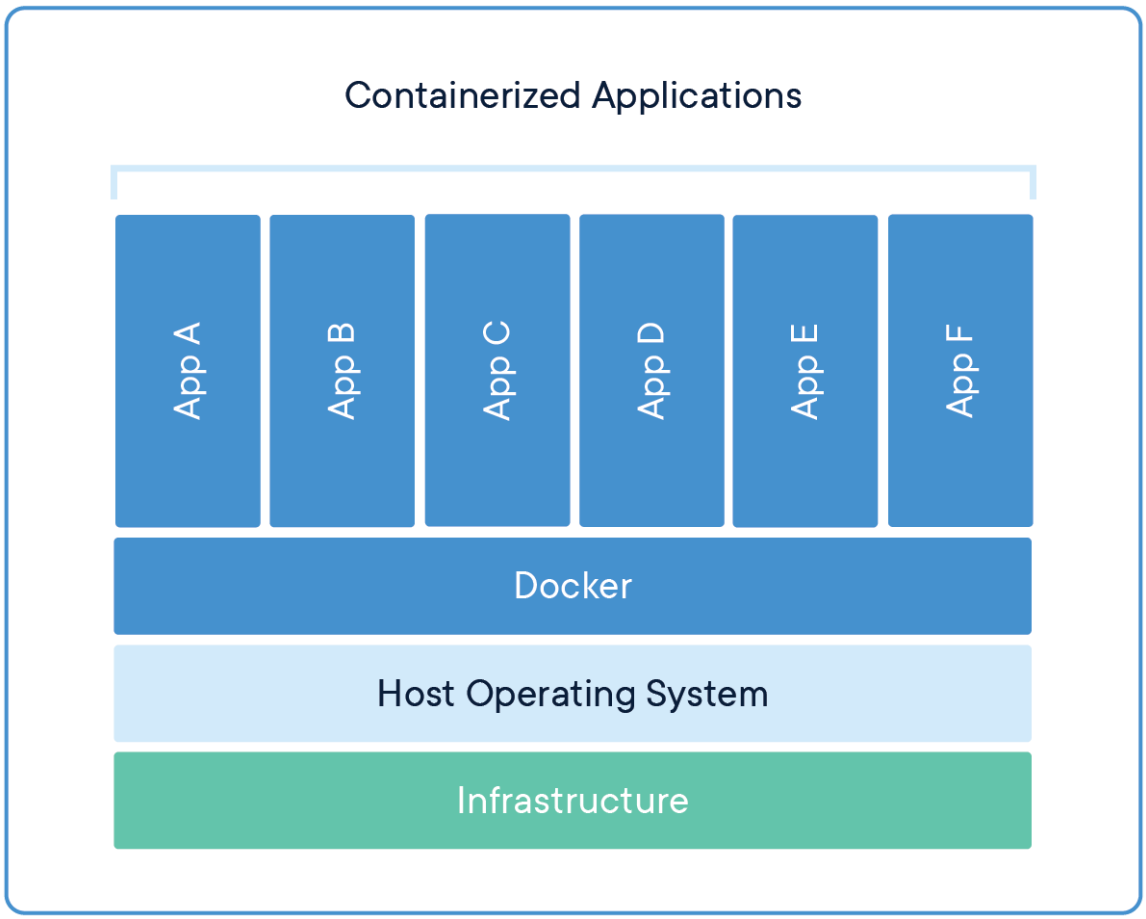


Рис. 4.6 – Принцип роботи контейнеру

Гіпервізор дозволяє на одній машині працювати декільком віртуальним машинам. Кожна віртуальна машина містить повну копію операційної системи, додаток, необхідні бінарні файли та бібліотеки через що займає десятки гігабайт. Також недоліком віртуальних машин є те, що вони можуть повільно стартувати.

У порівнянні, контейнери (рисунок 4.7) - це абстракція на рівні програми, яка пакує код і залежності разом. Кілька контейнерів можуть працювати на одній машині і обмінюватися ядром операційної системи з іншими контейнерами, кожен з яких працює як окремі процеси в просторі користувача. Контейнери займають менше місця, ніж віртуальні машини (зображення контейнерів зазвичай становлять десятки МБ), можуть обробляти більше додатків і потребують меншої кількості віртуальних машин та операційних

систем. Контейнери також мають можливість запускатися всередині віртуальних машин. В цілому використання контейнерів є більш вигідним у випадку коли необхідну конфігурувати середовище для виконання додатку, завдяки тому, що зібране зображення контейнеру вимагає набагато менше ресурсів та часу потрібного для старту порівняно з віртуальними машинами.

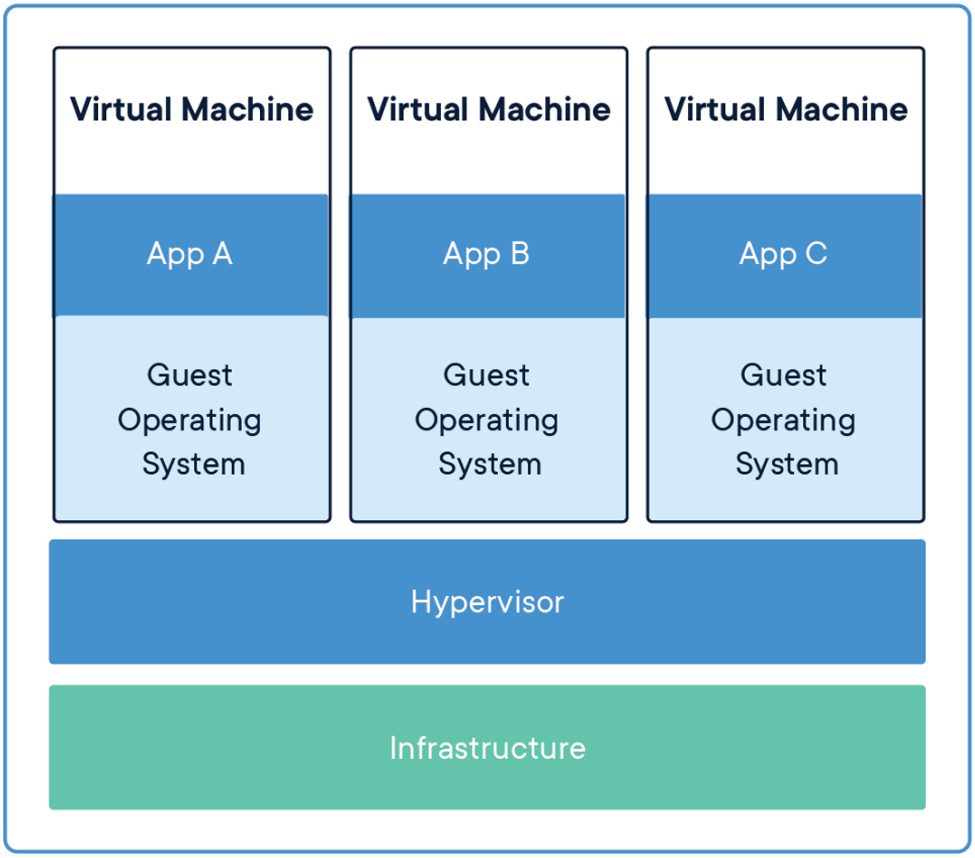


Рис. 4.7 – Принцип роботи віртуальної машини

Kubernetes в свою чергу це оркестратор який дозволяє керувати великою кількістю контейнерів та забезпечувати їх горизонтальне масштабування.

Сервіс Kubernetes Engine в свою чергу є абстракцією поверх Kuberntes яка дозволяє автоматично налаштовувати кластери Kubernetes та дозволяє горизонтально та вертикально масштабувати віртуальні машини кластеру для збільшення ресурсів необхідних для запуску додатків в контейнерах. Це дозволяє не витрачати велику кількість на часу на налаштування ресурсів неохідних для запуску Kubernetes. Хоча Kubernetes і надає велику кількість

переваг у порівнянні з іншими рішеннями запуску контейнерів, його складність є перешкодою для його використання.

Переваги:

- можливість запуску додатків в контейнері;
- автоматичне масштабування віртуальних машин.

Недоліки:

- складність технології та її підтримки.

4.2.5.3 App Engine

App Engine – повністю керована, безсерверна платформа для створення додатків. Дозволяє створювати, розгортати та масштабувати додатки не турбуючись про управління інфраструктурою. Дозволяє сконцентруватися на написанні коду, адже не потребує ніяких додаткових конфігурацій з боку серверу. Сервіс App Engine добре підходить для написання мікросервісних рішень, адже дозволяє запускати велику кількість сервісів в одному проекті. Сервіс підтримує два середовища виконання – Standard та Flexible.

В середовищі виконання Standard додатки запускаються в пісочниці яка підтримує наступні мови програмування – Python, Java, Node.js, PHP, Ruby та Go. Standard середовище краще підходить для випадків коли не очікується великого навантаження на додаток.

В середовищі Flexible додатки запускаються в контейнерах Docker в віртуальних машинах сервісу Compute Engine. Середовище краще підходить для випадків коли очікується постійне навантаження на додаток. Так як додатки працюють в контейнері, середовище підтримує будь які мови програмування які можуть працювати у конейнері Docker.

Основним недоліком сервісу є те що він створюється один на проект, і хоча і дозволяє створювати велику кількість сервісів але ці сервіси є тісно пов’язаними через що не вдається налаштувати різні рівні доступити до різних мікросервісів працюючих в середині App Engine. Це призводить до проблем з безпекою, адже якщо необхідно дати доступ користувачам до одного

мікросервісу, то досить важко надійно захистити інші.

Переваги:

- повністю кероване середовище для запуску;
- можливість запускати додатки всередині контейнерів.

Недоліки:

- неможливість налаштування вибіркового доступу до різних додатків в рамках одного проекту.

4.2.5.4 Cloud Run

Cloud Run - це керована обчислювальна платформа, яка дозволяє запускати контейнери, які можна викликати через веб-запити або події Pub / Sub. Cloud Run безсерверний: він відбирає все управління інфраструктурою, та дозволяє зосередитись на тому, що найбільше важливо - створенні додатків. Він створений на базі Knative, що дозволяє запускати контейнери або за допомогою Cloud Run, або в кластері Google Kubernetes Engine з Cloud Run for Anthos в Google Cloud [52].

Можливість легкого переходу між Cloud Run та Kubernetes є дуже корисною, адже дозволяє використовувати Cloud Run до тих пір поки сервіс задовольняє всі потреби, а у разі необхідності більш гнучкого налаштування легко перейти на Kubernetes.

Завдяки тому, що всі додатки в середині Cloud Run виконуються в контейнерах, він підтримує будь які мови програмування які підтримує виконуюче середовище контейнера. Також сервіс надає контроль доступу до кожного додатку який працює у рамках одного проекту, що дозволяє легко налаштовувати доступ ззовні для кожного сервісу.

Одним з недоліків сервісу є неможливість збереження даних в середовищі в якому виконується додаток у зв’язку з особливостями внутрішньої реалізації сервісу. Це викликає необхідність використовувати базу даних, кеш чи об’єктне сховище у випадку коли необхідно зберігати данні під час вмконання программи.

Переваги:

- повністю кероване середовище для запуску;
- налаштування вибіркового доступу до різних додатків в рамках одного проекту;
- легкий перехід у разі необхідності до сервісу Kubernetes.

Недоліки:

- неможливість збереження даних в середовищі де виконується додаток.

Cloud Run є найбільш вдалим варіантом, адже він дозволяє не витачати часу на його налаштування та в той же час надає великі можливості для масштабування. Тому для веб-серверу був обраний саме сервіс Cloud Run. Код веб-серверу а також Dockerfile необхідний для запуску на платформі Cloud Run знаходиться в додатку Б.

4.3 Висновки

Побудова платформи дозволить обробляти велику кількість даних що надходять з девайсів. Завдяки тому, що платформа побудована на базі Google Cloud Platform вона може витримувати велике навантаження та масштабуватися автоматично при необхідності. Використання сервісу Cloud IoT Core дозволяє гнучко налаштовувати роботу з кожним з підключених девайсів. Завдяки використанню сервісу BigQuery який створенний для роботи з великими даними, робота з даними девайсів які зберігаються в плафтормі є досить швидкою та простою. Також сервіс BigQuey надає великі можливості для інтеграції з сервісами обробки великих даних, таких як, наприклад, Apache Spark та Hadoop.

5 РОЗРОБКА МЕТЕОСТАНЦІЇ

5.1 Вступ

Для перевірки можливостей платформи розробляємо в даній роботі побудуємо пристрій метеостанції який буде здатний надсилати метрики до платформи за допомогою протоколу HTTP.

Будь-яка метеостанція містить в своєму складі датчик температури і повітря. Також обов'язковим для таких метеостанцій є датчик тиску. За допомогою датчика тиску можна визначати тенденції зміни погоди. Зараз, більшість домашніх метеостанцій роблять модульними, що дозволяє додавати до базових датчиків, датчики, які вимірюють додаткові метеорологічні величини. Комплект датчиків визначається при замовленні виробу і може включати в себе:

- датчик температури повітря;
- пірометричний датчик для безконтактного вимірювання температури поверхні будь-якого об'єкта;
- датчик вологості повітря;
- датчик атмосферного тиску;
- датчик швидкості вітру;
- датчик напрямку вітру;
- датчик кількості опадів;
- датчик рівня снігового покриву.

Проаналізовавши аналогічні рішення можна виділити спільні недоліки які мають майже всі проаналізовані рішення:

- можливість використання обмеженої кількості датчиків для одного користувача;
- необхідність наявності Ethernet підключення.

Так як проблема з кількістю датчиків для одного користувача

вирішується з боку самої платформи дистанційного контролю параметрів віддалених об’єктів і не потребує додаткових змін у самому пристрої сконцентруємося на проблемі необхідності наявності Ethernet підключення.

Це є проблемою у тих випадках коли пристрій знаходиться в місцях не звичайний провідний інтернет не є доступним, наприклад, у ліса чи полях. Для вирішення цієї проблеми в даній роботі для побудування пристрою метеостанції буде використовуватися мобільний інтернет Так як мобільний інтернет забезпечує значно більше покриття аніж провідний інтернет, це дозволить використовувати його в більш віддалених містах.

Необхідно зазначити, що хоч в пристрої метеостанції в даній роботі і буде використано мобільний інтернет, платформа не має жодних обмежень на тип інтернету і дозволяє надсилати інформації з приладів підключених за допомогою будь-якого типу інтернету.

5.2 Розробка структурної схеми приладу

Для побудування приладу метеостанції побудуємо його структурну схему (рисунок 5.1). Розглянемо основні компоненти зображені на структурній схемі.

Рисунок 5.1 – Структурна схема метеостанції

Основним компонентом приладу є мікроконтролер. Мікроконтролер – це виконана у вигляді мікросхеми спеціалізована мікропроцесорна система, що включає мікропроцесор, блоки пам'яті для збереження коду програм і даних, порти вводу-виводу і блоки зі спеціальними функціями (лічильники,

компаратори, аналого-цифровий перетворювач та інші). Використовується для керування електронними пристроями.

По суті, це — однокристальний комп'ютер, здатний виконувати прості завдання. Використання однієї мікросхеми значно знижує розміри, енергоспоживання і вартість пристроїв, побудованих на базі мікроконтролерів. Мікроконтролери можна зустріти в багатьох сучасних приладах, таких як телефони, пральні машини, вони відповідають за роботу двигунів і систем гальмування сучасних автомобілів, з їх допомогою створюються системи контролю і системи збору інформації.

Переважна більшість процесорів, що випускаються у світі — мікроконтролери [53]. Іноді їх називають вбудованим контролером або мікроконтролером (MCU), серед інших пристроїв мікроконтролери знаходяться в транспортних засобах, роботах, офісних машинах, медичних пристроях, мобільних радіоприймачах, торгових автоматах та побутовій техніці.

Мікроконтролер вбудований всередину системи для управління сингулярною функцією в пристрої. Це робиться шляхом інтерпретації даних, отриманих від периферійних пристроїв вводу / виводу за допомогою центрального процесора. Тимчасова інформація, яку отримує мікроконтролер, зберігається в її пам'яті даних, де процесор отримує доступ до неї і використовує інструкції, що зберігаються в її програмній пам'яті, для розшифровки та застосування вхідних даних.

Потім він використовує свою вхідно-вихідну периферію для зв'язку та здійснення відповідних дій. Мікроконтролери використовуються в широкому наборі систем і пристроїв. Пристрої часто використовують кілька мікроконтролерів, які працюють разом у межах пристрою для вирішення відповідних завдань. Наприклад, автомобіль може мати багато мікроконтролерів, які керують різними окремими системами всередині, такими як антиблокувальна гальмівна система, контроль тяги, управління впорскуванням палива або регулювання підвіски.

Всі мікроконтролери спілкуються між собою, щоб повідомити про

правильні дії. Деякі можуть спілкуватися зі складнішим центральним комп'ютером всередині автомобіля, а інші можуть спілкуватися лише з іншими мікроконтролерами. Вони надсилають та приймають дані за допомогою периферійних пристроїв вводу-виводу та обробляють ці дані для виконання призначених завдань.

Процесор (ЦП) - Процесор можна розглядати як мозок пристрою. Він обробляє і реагує на різні інструкції, які спрямовують функцію мікроконтролера. Це включає виконання основних арифметичних, логічних та операцій вводу / виводу. Він також виконує операції з передачі даних, які передають команди іншим компонентам більшої вбудованої системи.

Пам'ять - пам'ять мікроконтролера використовується для зберігання даних, які отримує процесор і використовує для відповіді на інструкції, запрограмовані для виконання. Мікроконтролер має два основних типи пам'яті:

– програма пам'яті, яка зберігає довгострокову інформацію про інструкції, які виконує процесор. Пам'ять програми - енергонезалежна пам'ять, тобто вона зберігає інформацію з часом, не потребуючи джерела живлення;

– пам'ять даних, яка потрібна для тимчасового зберігання даних під час виконання інструкцій. Пам'ять даних є мінливою, тобто дані, які вона зберігає, є тимчасовими і підтримуються лише в тому випадку, якщо пристрій підключено до джерела живлення.

Периферійні пристрої вводу/виводу - пристрої введення та виведення є інтерфейсом для процесора до зовнішнього світу. Порти введення отримують інформацію та надсилають її процесору у вигляді двійкових даних. Процесор отримує ці дані та надсилає необхідні вказівки на пристрої виводу, які виконують завдання, що знаходяться зовні мікроконтролера. Хоча процесор, пам'ять та периферійні пристрої вводу/виводу є визначальними елементами мікропроцесора, є й інші елементи, які часто включаються. Сам термін периферійні пристрої вводу/виводу просто відноситься до підтримуючих компонентів, які взаємодіють із пам'яттю та процесором. Існує багато допоміжних компонентів, які можна класифікувати як периферійні пристрої.

Наявність певного прояву периферійного вводу/виводу є елементом мікропроцесора, оскільки вони є механізмом, за допомогою якого застосовується процесор. Інші опорні елементи мікроконтролера включають:

– аналого-цифровий перетворювач (АЦП) - АЦП - схема, яка перетворює аналогові сигнали в цифрові сигнали. Це дозволяє процесору в центрі мікроконтролера взаємодіяти із зовнішніми аналоговими пристроями, такими як датчики;

– цифровий аналоговий перетворювач (ЦАП) - ЦАП виконує зворотну функцію АЦП і дозволяє процесору в центрі мікроконтролера передавати свої вихідні сигнали зовнішнім аналоговим компонентам;

– системна шина - це з'єднувальний провід, який з'єднує всі компоненти мікроконтролера разом;

– послідовний порт - послідовний порт - один із прикладів вводу-виводу, який дозволяє мікроконтролеру підключатися до зовнішніх компонентів. Він має функцію, подібну до USB або паралельного порту, але відрізняється тим, як обмінюється бітами.

Датчик вологості та температури повітря надає змогу вимірювати температуру та вологість за допомогою постійних вимірів та передачі інформації до мікроконтролерів. Частота виміру як знаходиться в пам’яті мікроконтролеру який в свою чергу надсилає команду до датчику та отримує поточні показники температури та вологості у відповідь.

Модуль GSM підключений до мікроконтролеру та дозволяє передавати данні отримані з датчику вологості та температури через мобільний інтернет за допомогою протоколу HTTP до платформи дистанційного контролю параметрів віддалених об’єктів. Під час включення пристрою мікроконтролер ініціює включення модулю GSM та його підключення до мобільної мережі.

5.3 Розробка пристрою метеостанції

Для побудування пристрою метеостанції який описаний структурною

схемою у попередньому розділі були обрані наступні компоненти. Мікроконтролер – Arduino Nano, Датчику вологості та температури – DHT11 та GSM модуль – SIM900. Структурна схема приладу показана на рисунку 5.2.

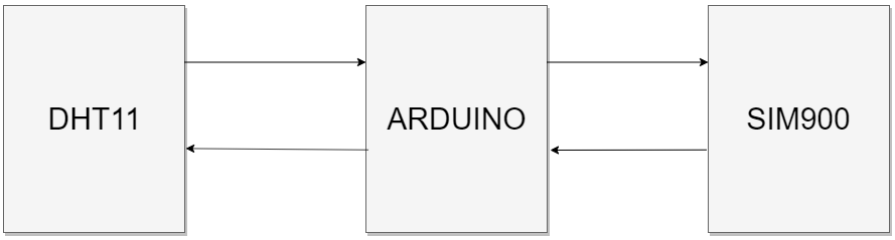


Рисунок 5.2 - Структурна схема пристрою метеостанції на базі Arduino

Arduino - торгова марка апаратно-програмних засобів для побудови простих систем автоматики і робототехніки. Програмна частина складається з програмної оболонки для написання програм, їх компіляції та програмування апаратури. Апаратна частина являє собою набір змонтованих друкованих плат, що продаються як офіційним виробником, так і сторонніми виробниками.

Arduino і Arduino-сумісні плати спроектовані таким чином, щоб їх можна було при необхідності розширювати, додаючи в пристрій нові компоненти. Ці плати розширень підключаються до Arduino за допомогою встановлених на них штирових роз'ємів.

Мікроконтролери для Arduino відрізняються наявністю попередньо прошитого в них завантажувача. За допомогою цього завантажувача користувач завантажує свою програму в мікроконтролер без використання традиційних окремих апаратних программаторов. Завантажувач з'єднується з комп'ютером через інтерфейс USB (якщо він є на платі) або за допомогою окремого перехідника UART-USB. Мовою програмування Arduino є стандартний C ++ (використовується компілятор AVR-GCC) з деякими особливостям [54].

Arduino – це платформа з відкритим кодом, заснована на простому у використанні апаратному та програмному забезпеченні. Протягом багатьох років Arduino був мозком тисяч проектів - від предметів побуту до складних наукових інструментів.

Всесвітня спільнота виробників - студентів, любителів, художників, програмістів та професіоналів - зібралася навколо цієї платформи з відкритим кодом, їх внески додали до неймовірної кількості доступних знань, які можуть бути дуже корисними як для початківців, так і для експертів. Arduino народився в Інституті дизайну взаємодії Ivrea як простий інструмент для швидкого складання прототипів, спрямований на студентів, що не мають досвіду електроніки та програмування.

Як тільки він дійшов до більш широкої спільноти, Arduino почав змінюватися, щоб адаптуватись до нових потреб та викликів, розмежуючи свою пропозицію від простих 8-бітних дошок до продуктів для IoT-додатків, носіїв, 3D-друку та вбудованих середовищ. Всі дошки Arduino мають повністю відкритим код, що дає можливість користувачам самостійно будувати їх і, зрештою, адаптувати їх під їх конкретні потреби. Програмне забезпечення теж є відкритим кодом, і воно зростає завдяки внеску користувачів у всьому світі [55].

Arduino Nano - це повнофункціональний мініатюрний пристрій на базі мікроконтролера ATmega328 (Arduino Nano 3.0) або ATmega168 (Arduino Nano 2.x), адаптоване для використання з макетної платою (рисунок 5.3).

За функціональністю пристрій схожий на Arduino Duemilanove, і відрізняється від нього розмірами, відсутністю роз'єму живлення, а також іншим типом (Mini-B) USB-кабелю. Arduino Nano розроблено і випускається фірмою Gravitech. Arduino Nano надає ряд можливостей для здійснення зв'язку з комп'ютером, ще одним Arduino або іншими мікроконтролерами. У ATmega168 і ATmega328 є приймач UART, що дозволяє здійснювати зв'язок з послідовним інтерфейсів за допомогою цифрових висновків 0 (RX) і 1 (TX).

Мікросхема FTDI FT232RL забезпечує зв'язок приймача з USB-портом комп'ютера, і при підключенні до комп'ютера дозволяє Arduino визначатися як віртуальний COM-порт. У пакет програмного забезпечення Arduino також входить спеціальна програма, що дозволяє зчитувати і відправляти на Arduino прості текстові дані [56].

SIM900 – це GSM модуль розроблений компанією SIMCom Wireless Solutions. Стандартний інтерфейс керування компонента SIM900 дозволяє отримувати доступ до мереж GSM/GPRS 850/900/1800/1900 МГц для використання дзвінків, надсилання СМС повідомлень та обміну цифровими данними через GPRS. Управляти модулем можна через персональний комп'ютер через інтерфейсу USB-UART чи безпосередньо через UART модуль Arduino, Raspberry Pi та аналогічних рішень.

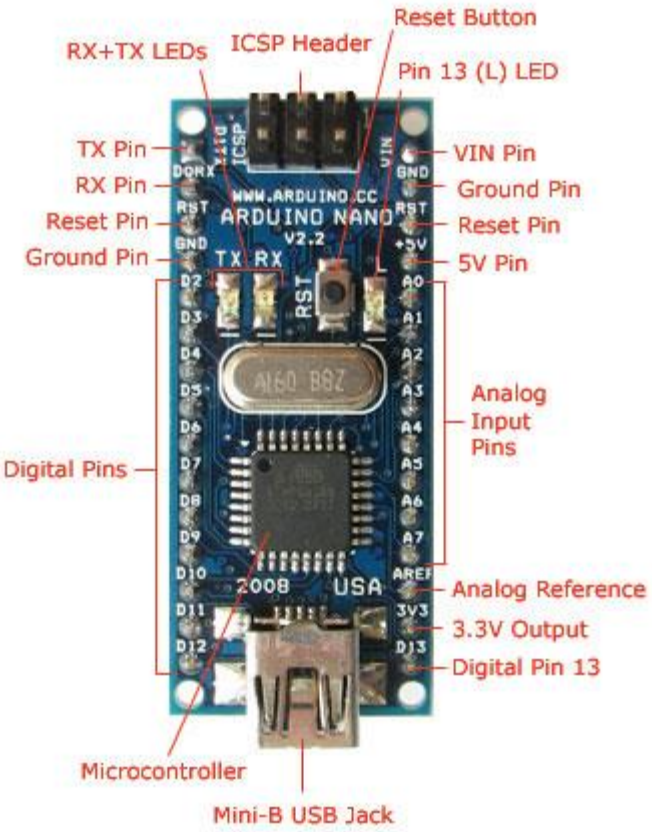


Рисунок 5.3 - Arduino Nano

Модуль DHT11 (рисунок 5.4) являє собою ємнісний датчик вологості і резистивний датчик температури, а також аналого-цифрового перетворювача, для перетворення аналогових вимірювань в цифровий вигляд і дозволяє проводити калібрування вихідного сигналу.

Після програмування мікроконтролеру Arduino програмним кодом який знаходиться в додатку А, підключення датчиків вологості та температури, а

також підключенні GSM модулі та підключенні пристрою до джерела живлення, пристрій має наступну логіку роботи:

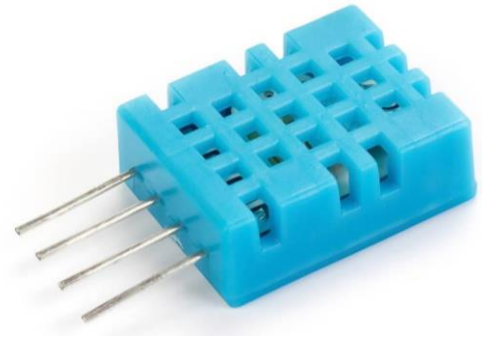


Рисунок 5.4 – Датчик DHT 11

– мікроконтролер посилає запит до модулю SIM900 для його включення;

– мікроконтроллер очікує доки модуль SIM900 включиться та встановить зв’язок з мережею;

– після отримання інформації про те, що модуль SIM900 встановив зв’язок з мережею, мікроконтролер починає циклічно опитувати датчик DHT11 та відправляти отримані з нього данні до платформи за допомогою модулю SIM900.

5.4 Висновки

Завдяки тому, що побудована метеостанція передає дані за допомогою GSM модулю, вона може бути використаною з любого місця де є мобільна мережа. Метеостанція дозволяє передавати температуру та вологість, але може віддавати і інші дані при підключені додаткових датчиків. Також метеостанція працює з побудованою платформою дистанційного контролю параметрів віддалених пристроїв та успішно надсилає до неї інформацію.

6 РОЗРОБКА ВЕБ-СЕРВЕРУ

Головна задача яку повинен вирішувати веб-сервер це надання користувачам можливості отримувати данні з їх приладів які підключені до платформи. Як було визначено раніше веб-сервер повинен працювати у сервісі Cloud Run. Для цього необхідно забезпечити можливість запуску веб-серверу в середені контейнеру, адже запуск додатку в середині контейнеру є єдиною можливість використання сервісу Cloud Run.

Серверний додаток написаний на мові програмування Go. Go — компільована мова програмування із вбудованими засобами для паралельних обчислень і засобами віддаленого керування пакунками. Цю мову програмування розробив Google як частину проекту з розробки операційної системи Inferno.

Початкова розробка Go почалася у вересні 2007 року, а безпосередньо проектували її Роберт Гризмер, Роб Пайк і Кен Томпсон. Офіційно мову представили у листопаді 2009 року. Метою створення проекту Go було бажання отримати мову, що поєднує високу продуктивність компільованих мов з легкістю написання коду, швидкістю розробки і захищеністю від помилок, властивих скриптовим мовам.

Синтаксис Go базується на звичних елементах мови Сі з окремими запозиченнями з мови Python. Мова досить лаконічна, але при цьому код залишається легким для читання і сприйняття. Проект від початку розробляється з оглядкою на багатониткове програмування та ефективну роботу на багатоядерних системах, в тому числі надаючи реалізовані на рівні операторів засоби для організації паралельних обчислень та взаємодії між паралельно виконуваними методами.

Мова надає вбудовані засоби захисту від виходу за допустимі області виділених блоків пам'яті і забезпечує можливість використання збирача сміття. При цьому код мовою Go компілюється у відокремлені бінарні виконувані файли, що виконуються напряму без використання віртуальної машини (модулі

профілювання та інші підсистеми виявлення проблем на етапі виконання інтегруються у вигляді runtime-компонентів), що дозволяє домогтися продуктивності порівнянної з програмами на мові Сі [57]. За словами Роба Пайка, «Go був розроблений для вирішення реальних проблем, що виникають при розробці програмного забезпечення в Google». В якості основних таких проблем він називає:

- повільну збірку програм;
- неконтрольовані залежності;
- використання різними програмістами різних підмножин мови;
- труднощі з розумінням програм, викликані складністю коду, поганим документуванням і так далі;
- дублювання розробок;
- високу вартість оновлень;
- несинхронні поновлення при дублюванні коду;
- складність розробки інструментарію;
- проблеми міжмовної взаємодії.

Передбачається, що програми на Go транслюватимуться компілятором в об'єктний код цільової апаратної платформи і надалі виконуватися безпосередньо, не вимагаючи віртуальної машини. Архітектура мови спочатку проектувалася так, щоб забезпечити швидку компіляцію в ефективний об'єктний код. Хоча для Go доступний і інтерпретатор, практично в ньому немає великої потреби, так як швидкість компіляції досить висока для забезпечення інтерактивної розробки.

Синтаксис мови Go схожий з синтаксисом мови C, з окремими елементами, запозиченими з мови Oberon і скриптових мов. Однією з основних сфер де використовується Go побудування веб-серверів, окрім цього Go також використовуються для побудування консольних додатків. Прикладом відомих рішень побудованих за допомогою Go є інструмент для роботи з контейнерами Docker та система керування контейнерами Kubernetes. Основними плюсами

якими володіє мова програмування Go є:

– компільованість – програма написана на Go компілюється в бінарний файл, що дозволяє виявляти помилки ще на етапі компіляції в порівнянні з інтерпретованими мовами програмування а також дозволяє виконувати скомпільовану програму без використання віртуальної машини;

– простий синтаксис – Go надає простий та стислий синтаксис для написання програм. Також мова надає єдиний правильний стиль оформлення коду, що запобігає різному написанню коду різними розробники та робить стиль написання коду уніфікованим;

– паралельність – Go підтримує написання паралельних програм, та надає засоби за допомогою яких написання паралельного коду є простими але в той же час надає великі можливості;

– статичне зв’язування – весь код програми статично пов’язується і при компіляції перетворюється в один бінарний файл;

– збирання сміття – Go має вистроєний збирач сміття, отже, управління пам'яттю відбувається автоматично, що зменшує вірогідність помилок які зазвичай можна зробити під час програмування на мовах які не мають власного збирача сміття і потребують ручного звільнення пам’яті.

Для викання програми в сервісі Cloud Run необхідно описати та збудувати зображення Docker контейнеру. Для побудування Docker контейнерів використовується консольний додаток Docker який будує зображення контейнеру на основі програмного коду а також файлу Dockerfile який містить в собі інструкції для побудування зображення контейнеру. Основними інструкціями які підтримує Dockrfile є:

– COPY – копіює данні з вказаної директорії чи конкретний файл до вказаної директорії всередині контейнеру;

– ADD - копіює данні з вказаної директорії чи конкретний файл до вказаної директорії всередині контейнеру. Також надає можливість вказувати архіви та посилання до ресурсів в мережі інтернет як вхідний ресурс та

отримувати розархівовні або завантажені з ресурсу данні в вказаній директорії в середині контейнеру;

– ENTRYPOINT – вказує вхідну точку яка починає виконуватися під час запуску контейнера;

– CMD – дозволяє вказувати додаткові параметри запуску;

– ENV – дозволяє встановити в контейнері змінні оточення;

– EXPOSE – дозволяє зв’язати порт контейнеру з портом машини на якій працює Docker для доступу до контейнеру ззовні;

– FROM – дозволяє вказати базове зображення контейнеру яке буде використовуватися під час побудування нового зображення;

– RUN – дозволяє виконувати операції та команди під час побудування зображення;

– USER – дозволяє встановити користувача з під якого будуть виконуватися програми та команди в контейнері

– VOLUME – дозволяє отримати доступ з контейнеру до директорії на машині на якій працює Docker;

– WORKDIR – дозволяє вказати директорію в якій будуть виконуватися команди в середині контейнеру;

– MAINTEINER – дозволяє вказати повне ім’я та адрес користувача який створив Dockerfile;

– Label – дозволяє додати власну позначку до зображення.

Файл Dockerfile для веб-серверу знаходиться в додатку В. Після того який зображення контейнеру було створену воно може бути завантажене до сервісу Cloud Run та використовуватися для створення нового сервісу.

Спілкування з веб-сервером відбувається за допомогою REST API на основі протоколу HTTP. REST – це архітектурний стиль та підхід до комунікації, який часто використовують під час розробки веб-серверів. Технології REST, як правило, віддається перевага чим більш надійній технології простого протоколу доступу до об'єктів (SOAP), оскільки REST

дозволяє використовувати меншу пропускну здатність, що робить її більш придатною для використання в Інтернеті. API для веб-сайту - це код, який дозволяє двом програмам спілкуватися між собою [58]. Технологія REST наразі є стандартом в розробці клієнт-серверних додатків.

Ключовим елементом в REST є ресурс. Ресурсом може бути, наприклад, динамічне значення чи документ. REST використовує типи повідомлень формату HTTP для побудування типу запиту. Стандарт HTTP має 8 типів повідомлень:

- GET – отримання стану ресурсу;
- POST – створення нового ресурсу;
- PUT – заміна поточного ресурсу на новий;
- DELETE – видалення ресурсу;
- PATCH – заміна частини ресурсу;
- HEAD – отримання заголовків без представлення;
- OPTIONS – отримання списку методів які підримує ресурс.

Властивості архітектури, які залежать від обмежень, накладених на REST-системи:

– продуктивність - взаємодія компонентів системи може бути домінуючим фактором продуктивності і ефективності мережі з точки зору користувача;

– маштабованість для забезпечення великої кількості компонентів і взаємодій компонентів.

Рой Філдінг - один з головних авторів специфікації протоколу HTTP, описує вплив архітектури REST на масштабованість наступним чином:

- простота уніфікованого інтерфейсу;
- відкритість компонентів до можливих змін для задоволення мінливих потреб (навіть при працюючому додатку);
- прозорість зав'язків між компонентами системи для сервісних служб;
- переносимість компонентів системи шляхом переміщення

програмного коду разом з даними;

– надійність, що виражається в стійкості до відмов на рівні системи при наявності відмов окремих компонентів, з'єднань, або даних.

Передача даних між сервером та клієнтом відбувається у форматі JSON. JSON (JavaScript Object Notation) - простий формат обміну даними, зручний для читання і написання як людиною, так і комп'ютером. Він заснований на підмножині мови програмування JavaScript, визначеного в стандарті ECMA-262 3rd Edition - December 1999. JSON - текстовий формат, повністю незалежний від мови реалізації, але він використовує угоди, знайомі програмістам C-подібних мов, таких як C, C ++, C # , Java, JavaScript, Perl, Python і багатьох інших. Ці властивості роблять JSON ідеальним мовою обміну даними.

JSON заснований на двох структурах даних:

– колекція пар ключ / значення. У різних мовах, ця концепція реалізована як об'єкт, запис, структура, словник, хеш, іменований список або асоціативний масив;

– упорядкований список значень. У більшості мов це реалізовано як масив, вектор, список або послідовність.

Це універсальні структури даних. Майже всі сучасні мови програмування підтримують їх в будь-якій формі. Логічно припустити, що формат даних, незалежний від мови програмування, повинен бути заснований на цих структурах[60]. JSON підтримує наступні типи даних:

– об’єкт;

– масив;

– рядок;

– число;

– булева змінна;

– Null.

Веб-сервер підтримує наступний прикладний програмний інтерфейс (API):

GET /metrics/{deviceID} – дозволяє отримати у відповідь інформацію про існуючи метрики для вказаного девайсу, де deviceID це ID активованого девайсу. Якщо записи для девайсу існують в платформі, то у відповідь приходить інформація про знайдені метрики у форматі:

```
{
    "id": {id},
    "deviceID": {deviceID},
    "data”: {data},
    "creationTime": {creationTime}
}
```

де id це id метрики, deviceID це id девайсу якому належить дана метрика, data це об’єкт в якому знаходиться інформація про метрики для даного запису а createTime це інформація про час створення метрики. Код веб-серверу знаходиться в додатку В.

Побудований веб-сервер дозволяє отримувати інформацію з обраного девайсу. За допомогою клієнтської бібліотеки BigQuery сервер здатен працювати з базою даних та вибирати необхідні для відповіді дані. Беручи до уваги факт, що веб-сервер розміщений в сервісі Cloud Run, веб-сервер має публічний адрес завдяки якому користувачи з підключенням до інтернету мають змогу з ним працювати.

ВИСНОВКИ

У данній роботі було розроблено платформу дистанційного контролю параметрів віддалених об’єктів. Для цього був виконаний аналіз аналогічних рішень та виявлені їх переваги та недоліки. На підставі їх недоліків а також поставленної задачі була розроблена структурна схема платформи. Також була розроблена реалізація структурної схеми за допогою Google Cloud Platform. Для побудування рішення був написаний веб-сервер на мові програмування Go.

Також, була розроблена структурна схема метеостанції. Метеостанція була побудова на базі Arduino. Метеостанція була підключена до розробленої платформи, що дозволило перевірити працездатність як самої метеостанції так і платформи.

Таким чином, усі сформульовані в роботі задачі були виконані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Інтернет речей [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82_%D1%80%D0%B5%D1%87%D0%B5%D0%B9 (Дата звернення 10.09.2019)

2. Розумний дім [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D1%83%D0%BC%D0%BD%D0%B8%D0%B9_%D0%B4%D1%96%D0%BC (Дата звернення 10.09.2019)

3. Великі данні [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%BB%D0%B8%D0%BA%D1%96_%D0%B4%D0%B0%D0%BD%D1%96 (Дата звернення 10.09.2019)

4. Способи використання Thingsboard [Електронний ресурс].- Режим доступу: https://thingsboard.io/iot-use-cases/ (Дата звернення 10.09.2019)

5. Рішення Thingsboard для збирання данних [Електронний ресурс].- Режим доступу: https://thingsboard.io/smart-metering/ (Дата звернення 10.09.2019)

6. Розумна енергетика за допомогою Thingsboard [Електронний ресурс].- Режим доступу: https://thingsboard.io/smart-energy/ (Дата звернення 10.09.2019)

7. Rule Engine платформи Thingsboard [Електронний ресурс].- Режим доступу: https://thingsboard.io/docs/user-guide/rule-engine-2-0/re-getting-started/ (Дата звернення 10.09.2019)

8. Початок роботи з Blynk [Електронний ресурс].- Режим доступу: https://blynk.io/en/getting-started (Дата звернення 12.09.2019)

9. Документація Blynk [Електронний ресурс].- Режим доступу: http://docs.blynk.cc/ (Дата звернення 12.09.2019)

10. Метеостанція за допомогою Blynk [Електронний ресурс].- Режим доступу: https://hackaday.io/project/159586-solar-powered-wifi-weather station/log/149099-interfacing-with-blynk-app (Дата звернення 12.09.2019)

11. Платформа myDevices [Електронний ресурс].- Режим доступу: https://mydevices.com/platform/ (Дата звернення 12.09.2019)

12. Датчик руху myDevices [Електронний ресурс].- Режим доступу: https://store.mydevices.com/product/radio-bridge-wireless-acceleration-based-movement-sensor/ (Дата звернення 12.09.2019)

13. Рішення для виміру температури myDevices [Електронний ресурс].- Режим доступу: https://mydevices.com/solutions/temperature/ (Дата звернення 12.09.2019)

14. Початок роботи з myDevices [Електронний ресурс].- Режим доступу: https://developers.mydevices.com/cayenne/docs/getting-started/ (Дата звернення 12.09.2019)

15. Дистанційний датчик метеостанції TFA 351133 Sun [Електронний ресурс].- Режим доступу: https://i.allo.ua/media/catalog/product/cache/1/image/425x295/799896e5c6c37e1160 8b9f8e1d047d15/t/f/tfa3511330121.jpg (Дата звернення 12.09.2019)

16. Дисплей базового блоку метеостанції TFA 351133 Sun - [Електронний ресурс].- Режим доступу: https://www.pollin.de/images/600x600x90/I590046.1-Funk-Wetterstation-mit-Farbdisplay-TFA-35-1133-01-Sun-schwarz.jpg (Дата звернення 13.09.2019)

17. Огляд метеостанції Oregon Scientific LW301 [Електронний ресурс].- Режим доступу: https://4pda.ru/forum/lofiversion/index.php? t795203.html (Дата звернення 13.09.2019)

18. Мобільний додаток метеостанції Oregon Scientific LW301 [Електронний ресурс].- Режим доступу: https://itunes.apple.com/us/app/os-anywhere-weather/id545866909?mt=8 (Дата звернення 13.09.2019)

19. Метеостанція La Crosse MA10050 [Електронний ресурс].- Режим доступу: http://prival.com.ua/images/catalog/meteosta ntsiya-la-crosse-ma10050 mobilniy-shlyuz-922438.jpg (Дата звернення 13.09.2019)

20. Мобільний додаток метеостанції La Crosse MA10050 [Електронний ресурс]. - Режим доступу: https://lh3.googleusercontent.com/

wrJyzLmNUl25ZRcHVUo0T6KmW2MXAuyLhXvNu5NvNdG96K26j96VopgXYZPsAEmo=h900 (Дата звернення 13.09.2019)

21. Метеостанція Netatmo Weather Station [Електронний ресурс].- Режим доступу: https://www.netatmo.com/product/weather/ (Дата звернення 13.09.2019)

22. Мобільний додаток метеостанції Netatmo Weather Station [Електронний ресурс].- Режим доступу: https://www.netatmo.com/ product/ weather/weatherstation (Дата звернення 13.09.2019)

23. Черга [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/%D0%A7%D0%B5%D1%80%D0%B3%D0%B0_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) (Дата звернення 10.10.2019)

24. HTTP [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/HTTP (Дата звернення 10.10.2019)

25. Огляд протоколу HTTP[Електронний ресурс].- Режим доступу: https://developer.mozilla.org/ru/docs/Web/HTTP/Overview (Дата звернення 10.10.2019)

26. Проксі[Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BA%D1%81%D1%96-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80 (Дата звернення 10.10.2019)

27. MQTT[Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/MQTT (Дата звернення 10.10.2019)

28. JWT[Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/JSON_Web_Token (Дата звернення 10.10.2019)

29. Вступ до JWT [Електронний ресурс].- Режим доступу: https://jwt.io/introduction/ (Дата звернення 10.10.2019)

30. PKCE [Електронний ресурс].- Режим доступу: https://auth0.com/docs/flows/concepts/auth-code-pkce (Дата звернення 10.10.2019)

31. RSA [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/RSA (Дата звернення 10.10.2019)

32. Google Cloud Platform [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/Google_Cloud_Platform (Дата звернення 10.10.2019)

33. Cloud Pub/Sub [Електронний ресурс].- Режим доступу: https://cloud.google.com/pubsub/docs/overview?hl=ru (Дата звернення 10.10.2019)

34. Модель публікаці-підписка [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/%D0%9F%D1%83%D0%B1%D0%BB%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F-%D0%BF%D1%96%D0%B4%D0%BF%D0%B8%D1%81%D0%BA%D0%B0_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) (Дата звернення 10.10.2019)

35. Безсерверні обчислення [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D1%96_%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F (Дата звернення 10.10.2019)

36. Мікросервіси [Електронний ресурс].- Режим доступу: https://habr.com/ru/post/249183/ (Дата звернення 15.10.2019)

37. BigQuery [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/BigQuery (Дата звернення 15.10.2019)

38. Геопросторові дані [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/%D0%93%D0%B5%D0%BE%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D1%80%D0%BE%D0%B2%D1%96_%D0%B4%D0%B0%D0%BD%D1%96 (Дата звернення 15.10.2019)

39. GeoJSON [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/GeoJSON (Дата звернення 15.10.2019)

40. MapReduce [Електронний ресурс].- Режим доступу: https://www.talend.com/resources/what-is-mapreduce/ (Дата звернення 25.10.2019)

41. Apache Spark [Електронний ресурс].- Режим доступу: https://spark.apache.org/ (Дата звернення 25.10.2019)

42. ETL [Електронний ресурс].- Режим доступу: https://www.ibm.com/support/knowledgecenter/en/SSWSR9_11.5.0/com.ibm.mdshs.initiateglossary.doc/topics/r_glossary_etl.html (Дата звернення 25.10.2019)

43. Вилучення, перетворення, завантаження (ETL) [Електронний ресурс].- Режим доступу: https://docs.microsoft.com/ru-ru/azure/architecture/data-guide/relational-data/etl (Дата звернення 25.10.2019)

44. Hadoop [Електронний ресурс].- Режим доступу: https://hadoop.apache.org/ (Дата звернення 25.10.2019)

45. Архітектура Hadoop YARN (Дата звернення 25.10.2019)

46. HDFS [Електронний ресурс].- Режим доступу: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html (Дата звернення 25.10.2019)

47. Hadoop MapReduce [Електронний ресурс].- Режим доступу: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html (Дата звернення 25.10.2019)

48. Dremel MapReduce [Електронний ресурс].- Режим доступу: http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36632.pdf (Дата звернення 25.10.2019)

49. Внутрішня реалізація BigQuery [Електронний ресурс].- Режим доступу: https://cloud.google.com/blog/products/gcp/bigquery-under-the-hood (Дата звернення 10.11.2019)

50. Контейнер [Електронний ресурс].- Режим доступу: https://www.docker.com/resources/what-container (Дата звернення 10.11.2019)

51. Docker [Електронний ресурс].- Режим доступу: https://aws.amazon.com/en/docker/ (Дата звернення 10.11.2019)

52. Cloud Run [Електронний ресурс].- Режим доступу: https://cloud.google.com/run/docs/ (Дата звернення 10.11.2019)

53. Мікроконтроллер [Електронний ресурс].- Режим доступу: http://wiki.kubg.edu.ua/%D0%9C%D1%96%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80 (Дата звернення 20.11.2019)

54. Arduino [Електронний ресурс].- Режим доступу: https://ru.wikipedia.org/ wiki/Arduino (Дата звернення 20.11.2019)

55. Вступ до Arduino [Електронний ресурс].- Режим доступу: https://www.arduino.cc/en/Guide/Introduction (Дата звернення 20.11.2019)

56. Arduino Nano [Електронний ресурс].- Режим доступу: https://doc.arduino. ua/ru/hardware/Nano (Дата звернення 20.11.2019)

57. Мова програмування Go [Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/Go_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) (Дата звернення 20.11.2019)

58. REST [Електронний ресурс].- Режим доступу: https://searchapparchitecture.techtarget.com/definition/RESTful-API (Дата звернення 20.11.2019)

59. Властивості архітектури REST [Електронний ресурс].- Режим доступу: https://ru.wikipedia.org/wiki/REST (Дата звернення 30.11.2019)

60. JSON [Електронний ресурс].- Режим доступу: https://www.json.org/json-ru.html (Дата звернення 30.11.2019)

61. Програмна модель MapReduce[Електронний ресурс].- Режим доступу: https://uk.wikipedia.org/wiki/MapReduce (Дата звернення 30.11.2019)