ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:«Розробка системи обробки та моніторингу MQTT повідомлень на основі Node-Red, InfluxDB і Grafana»

на здобуття освітнього ступеня магістра

зі спеціальності 126 Інформаційні системи та технології

(код, найменування спеціальності)

освітньо-професійної програми 126 Інформаційні системи та технології

(назва)

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

_____ Вашкулат Кирило
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти Вашкулат Кирило
група ІСДМ-62

Керівник: Ігор Сініцин
науковий ступінь, вчене звання к.т.н., професор

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

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедрою ІПЗАС

_____ Каміла СТОРЧАК

«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Вашкулат Кирило Сергійович

(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка системи обробки та моніторингу MQTT повідомлень на основі Node-Red, InfluxDB і Grafana

керівник кваліфікаційної роботи Ігор Сініцин КТН Професор,

(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.

3. Вихідні дані до кваліфікаційної роботи:

Наукова-технічна література; Нормативні матеріали

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд Node-Red, InfluxDB і Grafana;

2. Встановлення та налаштування Docker контейнера;

3. Розгортання InfluxDB та Grafana

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Огляд Node-Red, InfluxDB і Grafana	19.10-05.11.23	Виконано
2	Протоколи та інструменти обміну iot даними	05.11-12.11.23	Виконано
3	Детальний огляд веб-потоків	13.11-19.11.23	Виконано
4	Використання docker та контейнерів для обробки iot даних	20.11-25.11.23	Виконано

5	Встановлення та налаштування Docker контейнера	27.11-03.12.23	Виконано
6	Налаштування вузлів Node-RED	04.12-10.12.23	Виконано
7	Розгортання InfluxDB та Grafana	11.12-20.12.23	Виконано
8	Вступ, висновок,	21.12-29.12.23	Виконано

Здобувач(ка) вищої освіти _____ Кирило Вашкулат

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник

кваліфікаційної роботи _____ Ігор Сініцин

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

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

Направляється здобувач Вашкулат К.С. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 126 Інформаційні системи та технлогії
(код, найменування спеціальності)
освітньо-професійної програми 126 Інформаційні системи та технлогії
(назва)
на тему: «Розробка системи обробки та моніторингу MQTT повідомлень на основі Node-Red, InfluxDB і Grafana». Кваліфікаційна робота і рецензія додаються.

Директор ННІ _____ Бондарчук А.П.
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач(ка) дослідив та описав використання Docker застосунків для обробки IoT даних. Розгялнута важливість та актуальність використання протоколів передачі даних як MQTT, HTTP та Websocket. Для обробки даних були використанні застосунки MQTT брокер, InfuxDB та Grafana. В якості апаратного забезпечення були використанні одноплатний комп'ютер Raspberry Pi та мікроконтролери ESP8266/32 з сенсором BME280

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача Вашкулата Кирила Сергійовича на оцінку «відмінно» та присвоїти йому кваліфікацію _____.

Керівник кваліфікаційної роботи _____ Сініцин І.П.

(підпис) (Ім'я, ПРІЗВИЩЕ)

«____» _____ 20___ року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач Вашкулат К.С. допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою ІПЗАС _____ Сторчак К.П.

(назва) (підпис) (Ім'я, ПРІЗВИЩЕ)

ВІДГУК РЕЦЕНЗЕНТА

на кваліфікаційну магістерську роботу

здобувача(ки) вищої освіти Вашкулат Кирило Сергійович

(прізвище, ім'я, по батькові)

на тему «Розробка системи обробки та моніторингу MQTT повідомлень на основі Node-Red, InfluxDB і Grafana»

Актуальність.

Робота є актуальною у зв'язку зі зростанням інтересу до Інтернету Речей та потреби в ефективних інструментах обробки та моніторингу даних з різних джерел. Розроблена система на базі Node-Red, InfluxDB і Grafana відповідає цим вимогам, забезпечуючи можливість легкої інтеграції, візуалізації та аналізу MQTT даних, що робить її корисною для широкого спектру застосувань в галузі IoT та моніторингу систем.

Позитивні сторони.

1. Для перевірки запропонованих концепцій були використані симуляційні та реальні дані.

2. Порівнювалися різні вузли MQTT брокерів.

3. Запропоновані ідеї були перевірені на практиці.

Недоліки.

1. Не розглянуто споживання системних ресурсів запропонованими системами.
2. Не розглянуто вартість системи.

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної магістерської роботи.

Висновок: *кваліфікаційна магістерська робота заслуговує оцінку "відмінно", а здобувач* Вашкулат К.С. *заслуговує присвоєння кваліфікації:*

.....

Рецензент: _____ _____

науковий ступінь, вчене звання *підпис* *Ім'я, ПРІЗВИЩЕ*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 76 стор., 51 рис., 1 табл., 50 джерел.

Мета роботи – розробка та вивчення системи обробки та моніторингу MQTT повідомлень на базі Node-Red, InfluxDB і Grafana з метою створення ефективного інструменту для контролю та візуалізації даних в інтернеті речей.

Об'єкт дослідження – система обробки та моніторингу MQTT повідомлень на базі Node-Red, InfluxDB і Grafana, яка представляє собою комплексне явище в галузі інтернету речей.

Предмет дослідження – аспекти розробленої системи, такі як її ефективність, простота розробки, масштабованість, візуалізація даних, відкритий код тощо, що визначається як основні характеристики та функціональність.

Короткий зміст роботи: Дипломна робота присвячена розробці та дослідженню системи обробки та моніторингу MQTT повідомлень, що передаються з ESP8266 та сенсора BME280 на основі Node-Red, InfluxDB і Grafana для забезпечення ефективного контролю та візуалізації даних в інтернеті речей.

КЛЮЧОВІ СЛОВА: IoT, ESP8266, ESP32, GRAFANA, INFLUXDB, NODE-RED, DOCKER

ABSTRACT

Text part of the master's qualification work:
76 pages, 51 pictures, 1 table, 50 sources.

The purpose of the work - development and study of the MQTT message processing and monitoring system based on Node-Red, InfluxDB and Grafana in order to create an effective tool for monitoring and visualizing data in the Internet of Things.

Object of research – a system for processing and monitoring MQTT messages based on Node-Red, InfluxDB and Grafana, which is a complex phenomenon in the field of the Internet of Things.

Subject of research – aspects of the developed system, such as its efficiency, ease of development, scalability, data visualization, open source, etc., defined as core features and functionality.

Summary of the work - The thesis is devoted to the development and research of a system for processing and monitoring MQTT messages transmitted from ESP8266 and

the BME280 sensor based on Node-Red, InfluxDB and Grafana to ensure effective control and visualization of data in the Internet of Things.

KEYWORDS: IoT, ESP8266, ESP32, GRAFANA, INFLUXDB, NODE-RED, DOCKER

ЗМІСТ

ВСТУП

Інтернет речей (IoT) та домашня автоматизація відіграють важливу роль у технологічному ландшафті, забезпечуючи зручність та ефективність. IoT - це мережа з'єднаних пристроїв, які обмінюються даними через Інтернет. У домашній автоматизації IoT створює інтелектуальне середовище, де пристрої автоматизовано взаємодіють для полегшення повсякденних завдань. Смарт-пристрої та мобільні додатки дозволяють власникам дистанційно керувати освітленням, опаленням, безпекою та енергозбереженням. Ці технології підвищують комфорт, енергоефективність та безпеку в домі, перетворюючи наше оточення та взаємодію з ним.

Але для того, щоб контролювати та контролювати їх, необхідно мати панелі керування та платформу, яка виконує всю інтеграцію. Цей дипломний проект інтегрує данні від сенсорів в Node-RED і власну панель керування, зберігає їх в базі даних InfluxDB і візуалізує їх у Grafana. Також було досліджено концепцію Docker контейнерів для використання у домашній автоматизації та IoT.

1 ПРОТОКОЛИ ТА ІНСТРУМЕНТИ ОБМІНУ ІОТ ДАНИМИ

MQTT (Message Queuing Telemetry Transport) - це протокол передачі повідомлень, який був розроблений для обміну даними між пристроями та застосунками в мережах IoT (Internet of Things) та M2M (Machine-to-Machine) [1]. MQTT став особливо популярним у світі IoT завдяки своїй легкості використання та ефективності передачі даних.

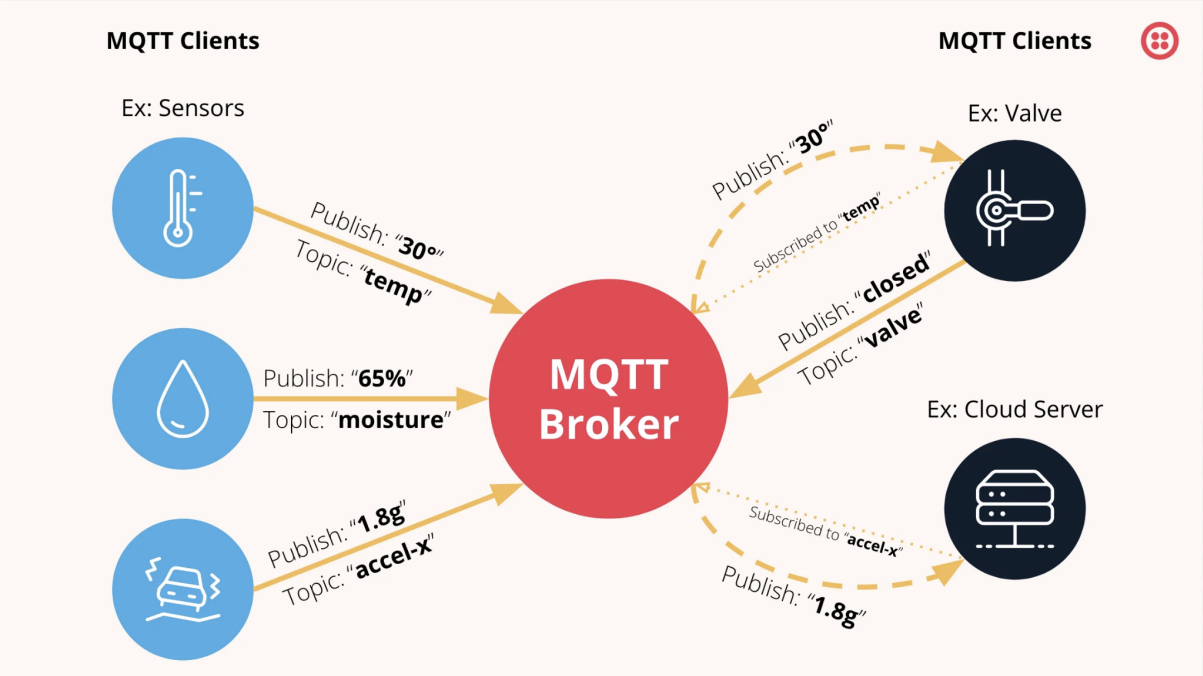


Рисунок 1.1 Принцип роботи протоколу MQTT

Важливість MQTT:

- Легкість використання: MQTT призначений для ефективного обміну повідомленнями та вимагає мінімальних зусиль для налаштування та використання. Це робить його ідеальним протоколом для взаємодії між різними IoT-пристроями, навіть з обмеженими ресурсами.
- Мінімальна пропускна здатність: MQTT ефективно використовує мережевий трафік завдяки своєму легковажкому протоколу. Це важливо у випадках, коли ресурси мережі обмежені або коштують дорого.
- Стійкість до втрати з'єднання: MQTT може відновлювати з'єднання в разі втрати зв'язку, що робить його надзвичайно надійним для застосунків IoT, де пристрої можуть бути тимчасово недоступними.

- Підтримка якісного обслуговування обслуговування (QoS): MQTT підтримує рівні якості обслуговування від 0 (найменший рівень гарантії доставки) до 2 (найвищий рівень гарантії доставки). Це дозволяє вибирати оптимальний рівень залежно від ваших потреб.
- Широкий застосунок: MQTT використовується в різних галузях, таких як домашній автоматизації, транспорт, сільське господарство, медицина та багато інших. Він дозволяє підключати мільйони пристроїв і збирати дані для аналізу та керування.
- Відкритий стандарт: MQTT є відкритим стандартом з відкритими специфікаціями, що сприяє його відкритому розвитку та інтероперабельності між різними реалізаціями.
- Підтримка публікації-підписки: MQTT використовує модель публікації-підписки, що дозволяє пристроям відправляти повідомлення (публікація) та підписуватися на отримання певних типів повідомлень (підписка) за допомогою тематичних каналів. Це забезпечує розподілену комунікацію та зменшує навантаження на мережу.
- Масштабованість: MQTT може легко масштабуватися для підтримки великої кількості підключених пристроїв та мережевих вузлів. Це робить його ідеальним для великих систем IoT.
- Підтримка SSL/TLS для захисту даних: MQTT може використовувати SSL/TLS для шифрування трафіку та забезпечення безпеки даних, що передаються по мережі.

Усі ці особливості роблять MQTT дуже важливим протоколом для забезпечення ефективного обміну даними та моніторингу в мережах IoT та M2M. Він дозволяє створювати надійні та ефективні системи для збору, обробки та візуалізації даних, що є важливим завданням у сучасному світі зрослих об'ємів даних та підключених пристроїв.

Огляд ключових понять та характеристик MQTT

MQTT (Message Queuing Telemetry Transport) - це легкий протокол для обміну повідомленнями, який використовується для комунікації між пристроями

та додатками в мережах IoT (Internet of Things) та M2M (Machine-to-Machine)[2]. Ось огляд ключових понять та характеристик MQTT:

Брокер MQTT (MQTT Broker): Брокер MQTT - це посередник, який приймає повідомлення від відправників і розподіляє їх до приймачів на основі тематичних каналів. Він керує процесом реєстрації пристроїв, публікації та підписки на повідомлення.

Публікація (Publish): Публікація - це процес відправки повідомлення від відправника до брокера MQTT на певний тематичний канал. Після цього брокер розподіляє повідомлення всім підписаним на цей канал приймачам.

Підписка (Subscribe): Підписка - це процес реєстрації приймача на певний тематичний канал для отримання повідомлень, які публікуються на цьому каналі. Підписаний приймач отримує всі повідомлення, які надходять на канал.

Тематичний канал (Topic): Тематичний канал - це логічний шлях, за яким публікуються та приймаються повідомлення. Він визначає, кому призначене повідомлення та який приймач має його отримати. Прикладом тематичного каналу може бути "sensors/temperature" для передачі даних про температуру.

Якість обслуговування (Quality of Service, QoS): MQTT підтримує три рівні QoS, які визначають рівень надійності доставки повідомлень. Вони включають:

QoS 0 (At most once): Гарантує, що повідомлення можуть бути доставлені найменшій кількості разів.

QoS 1 (At least once): Гарантує, що повідомлення буде доставлено щонайменше один раз, але може бути доставлено більше одного разу.

QoS 2 (Exactly once): Гарантує, що повідомлення буде доставлено точно один раз.

Відкритий стандарт: MQTT є відкритим стандартом з відкритою специфікацією, що сприяє інтероперабельності між різними реалізаціями протоколу та сприяє розвитку екосистеми MQTT.

Стійкість до втрати з'єднання: MQTT має вбудовану стійкість до втрати з'єднання, що дозволяє пристроям відновлювати підключення до брокера після тимчасової втрати зв'язку.

Масштабованість: MQTT дозволяє підключати велику кількість пристроїв до брокера та легко масштабуватися для обробки великого обсягу даних.

Використання в IoT та M2M: MQTT дуже популярний у сферах IoT та M2M завдяки своїй легкості використання, ефективності та підтримці низькопотужних пристроїв.

MQTT є потужним інструментом для створення надійних та ефективних систем комунікації для IoT та M2M додатків, де ефективність, легкість використання та надійність грають важливу роль.

Аналіз сучасних вимог до систем обробки та моніторингу MQTT повідомлень

[3] Аналіз сучасних вимог до систем обробки та моніторингу MQTT повідомлень важливий для розуміння, які вимоги ставляться до таких систем у сучасному світі. Ось деякі з ключових вимог та тенденцій:

Складність інтеграції з IoT пристроями: З ростом кількості підключених IoT-пристроїв з'явилася потреба в простих та ефективних засобах інтеграції з цими пристроями. Системи повинні підтримувати різні пристрої та масштабуватися для обробки великої кількості підключень.

Підтримка великих обсягів даних: Системи моніторингу та обробки MQTT повідомлень повинні бути готові для обробки великих обсягів даних, особливо коли мова йде про системи IoT, де генерується велика кількість даних з сенсорів та пристроїв.

Забезпечення стійкості до втрати даних: Для багатьох застосунків критично важливо забезпечити стійкість до втрати даних. Це означає, що системи повинні мати можливість зберігати дані в надійному сховищі та відновлювати в разі втрати з'єднання або відмови пристроїв.

Підтримка масштабованості: Оскільки кількість підключених пристроїв може зростати дуже швидко, системи повинні бути легко масштабовані для взаємодії з великою кількістю пристроїв та обробки даних.

Реальний час та низька затримка: Деякі застосунки IoT вимагають обробки даних в реальному часі та мають низькі вимоги до затримки. Системи моніторингу повинні бути готові для реалізації цих вимог.

Візуалізація та аналіз даних: Користувачі вимагають інструменти для візуалізації та аналізу зібраних даних. Це допомагає зрозуміти стан системи та приймати обґрунтовані рішення.

Безпека даних: Захист даних є надзвичайно важливим, особливо в IoT-системах, де збираються важливі дані. Системи повинні включати в себе механізми автентифікації, шифрування та контролю доступу.

Можливість інтеграції з іншими системами: Важливо мати можливість інтегрувати системи моніторингу та обробки MQTT повідомлень з іншими інформаційними системами, що використовуються в організації.

Підтримка резервного копіювання і відновлення: Наявність механізмів резервного копіювання та можливості відновлення даних важлива для забезпечення безпеки та стійкості системи.

Ці вимоги стають все більш актуальними, оскільки використання MQTT та систем моніторингу для обробки даних зростає в IoT, великих даних та інших сферах. Вирішення цих вимог допомагає створити надійні та ефективні системи для обробки та моніторингу MQTT повідомлень.

1.1 Огляд Node-Red, InfluxDB і Grafana як інструментів для створення системи моніторингу та обробки даних

[4] Node-Red, InfluxDB, і Grafana - це потужні інструменти, які спільно використовуються для створення систем моніторингу та обробки даних у сферах Інтернету речей (IoT), великих даних та аналітики. Нижче наведено огляд кожного інструмента:

Node-Red - це візуальний інструмент для програмування потоків даних. Він дозволяє користувачам легко створювати, з'єднувати та керувати вузлами

(компонентами), які представляють операції обробки даних. Node-Red базується на JavaScript та використовується для розробки реактивних програм для обробки даних.

Node-Red ідеально підходить для збору, фільтрації, перетворення та розподілу даних, включаючи дані з IoT-пристроїв. Він також може використовуватися для взаємодії з різними інтерфейсами API і веб-службами.

Переваги: Легкість використання, візуальний інтерфейс, багатий набір готових вузлів, гнучкість та розширюваність.

InfluxDB - це відкрита, розподілена система для зберігання часових рядів даних. Вона оптимізована для зберігання, запитів та аналізу даних, які змінюються з часом. InfluxDB використовує мову запитів InfluxQL для роботи з даними.

InfluxDB добре підходить для зберігання та вивчення часових рядів даних, таких як дані сенсорів IoT, логи серверів, дані про продуктивність та багато інших. Він підтримує великі обсяги даних та має можливості для розширення.

Переваги: Висока швидкість запису/читання, підтримка гнучких часових інтервалів, можливість групування та агрегації даних.

Grafana - це платформа для візуалізації даних та моніторингу, яка надає можливість створювати інтерактивні та красиві графіки, графіки та панелі для відображення даних з різних джерел. Вона підтримує відображення реального часу та аналіз даних.

Grafana використовується для створення моніторингових панелей та графіків для відображення даних, які зберігаються у часових рядах (наприклад, дані з InfluxDB). Вона широко використовується для моніторингу систем, застосунків та Інтернету речей.

Переваги: Легкість налаштування та використання, підтримка різних джерел даних, розширюваність, можливість відображення в реальному часі, широкий спектр графічних можливостей.

Разом ці три інструменти (Node-Red, InfluxDB та Grafana) дозволяють легко створювати повнофункціональні системи моніторингу та обробки даних для IoT, великих даних та інших галузей, спрощуючи розробку, зберігання та відображення

даних. Вони часто використовуються в комплексі для створення повноцінних рішень для аналізу та візуалізації даних.

Важливість розробки системи на базі Node-Red, InfluxDB і Grafana

Розробка системи на базі Node-Red, InfluxDB і Grafana має величезну важливість у сучасному світі, особливо в контексті Інтернету речей (IoT), моніторингу та обробки даних. Нижче розглянемо важливі аспекти цієї розробки:

Легкість розробки та швидкість впровадження: Node-Red надає інтуїтивно зрозумлий візуальний інтерфейс для створення потоків даних без необхідності глибокого програмування. Це дозволяє розробникам швидко створювати та розгортати системи моніторингу та обробки даних, скорочуючи час до ринку.

Ефективна обробка та інтеграція даних: Node-Red дозволяє інтегрувати різні джерела даних, включаючи MQTT-пристрої, і оброблювати ці дані в реальному часі. Це важливо для збору та обробки інформації з багатьох джерел.

Зберігання та управління даними часових рядів: InfluxDB спеціалізується на зберіганні та оптимізованому запитуванні даних часових рядів, що робить його ідеальним для зберігання даних, зібраних з сенсорів та пристроїв IoT.

Візуалізація та аналіз даних: Grafana надає потужний інструмент для візуалізації даних та створення моніторингових панелей. Це допомагає операторам та аналітикам відстежувати стан системи в реальному часі та аналізувати зібрані дані.

Реальний час та масштабованість: Розробка на базі цих інструментів дозволяє створювати системи, які здатні працювати в реальному часі та масштабуватися для взаємодії з великою кількістю пристроїв та джерел даних.

Безпека та надійність: Гарантована безпека та надійність даних є важливою вимогою для систем моніторингу та обробки даних. Ці інструменти надають засоби для забезпечення захисту даних та стійкості системи.

Широкий спектр застосувань: Системи, розроблені на базі Node-Red, InfluxDB і Grafana, можуть застосовуватися в різних галузях, включаючи моніторинг виробництва, управління будівлями, сільське господарство, медицину, транспорт та інші.

Відкритий стандарт та спільнота: Ці інструменти базуються на відкритих стандартах і мають активні спільноти користувачів і розробників, що сприяє розвитку та підтримці цих інструментів.

Усе це робить розробку систем на базі Node-Red, InfluxDB і Grafana надзвичайно важливою для ефективного моніторингу, обробки та аналізу даних, особливо в галузі IoT та великих даних.

Вивчення функціональності та можливостей Node-Red

[5] Node-RED - це візуальний інструмент для програмування потоків даних, який дозволяє вам створювати розумні потоки обробки та маніпуляції даними в інтерактивному режимі. Ось огляд функціональності та можливостей Node-RED:

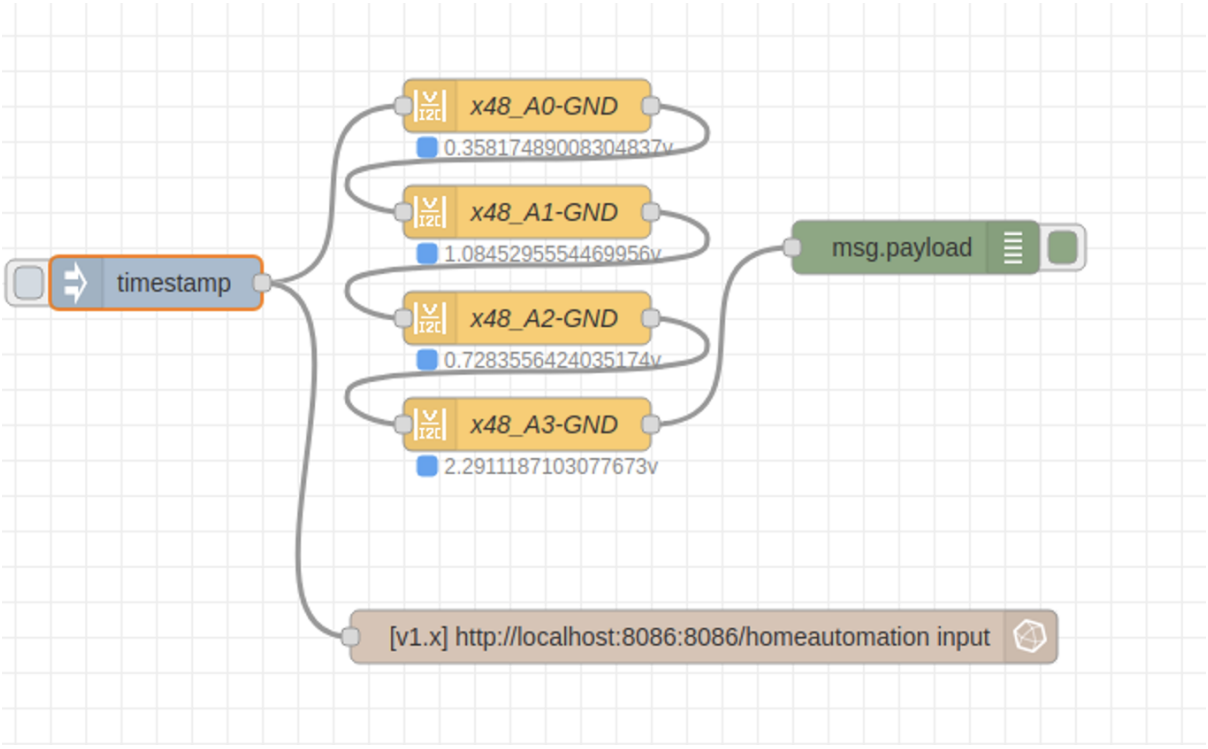


Рисунок 1.2 Приклад потоку Node-RED

Візуальний програмувальний інтерфейс: Node-RED надає візуальний інтерфейс, який дозволяє вам створювати та редагувати потоки даних за допомогою вузлів та з'єднань. Ви можете перетягувати та вставляти вузли для створення вашого потоку.

Багато готових вузлів: Node-RED поставляється з великою кількістю готових вузлів, які представляють різні операції обробки даних, включаючи роботу з файлами, роботу з мережею, візуалізацію даних та багато інших.

Підтримка власних вузлів: Ви можете створювати власні вузли та розширювати функціональність Node-RED, писуючи код на JavaScript або використовуючи інші мови програмування.

Інтеграція з різними джерелами даних: Node-RED підтримує інтеграцію з різними джерелами даних, включаючи бази даних, сенсори IoT, веб-служби, API, MQTT, Twitter, і багато інших. Ви можете легко отримувати та відправляти дані з різних джерел.

Розподілене виконання: Node-RED підтримує розподілене виконання потоків даних на вузли, що дозволяє використовувати багато ядер процесора та роздільні сервери для обробки великого обсягу даних.

Візуалізація даних: Ви можете використовувати вбудовані вузли для створення графіків, графіків та інших візуальних представлень ваших даних.

Можливість реакції на події: Node-RED дозволяє створювати потоки даних, які реагують на події, такі як зміна даних або вхід від користувача. Ви можете легко налаштовувати автоматизацію та обробку подій.

Підтримка розширень та інтеграція з сторонніми рішеннями: Node-RED може легко інтегруватися з іншими рішеннями та підтримує розширення за допомогою додаткових пакетів.

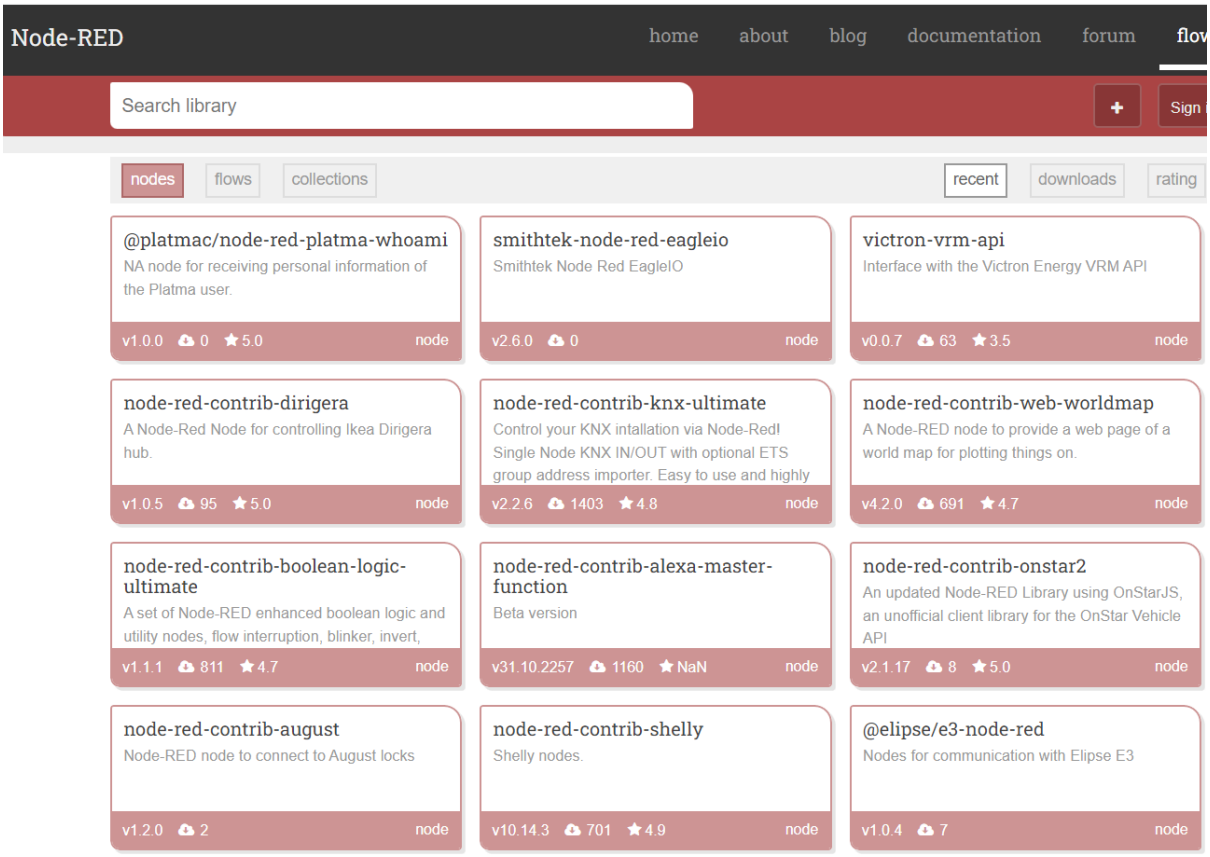


Рисунок 1.3 Бібліотека вузлів Node-RED

Локальна або хмарна розгортка: Ви можете розгортати Node-RED локально на вашому комп'ютері або в хмарному середовищі.

Спільнота та підтримка: Node-RED має активну спільноту користувачів та широкий обсяг документації, що дозволяє отримати підтримку та поради від інших користувачів.

Node-RED є потужним інструментом для розробки та виконання розумних потоків обробки даних, автоматизації та інтеграції різних систем, що дозволяє вам швидко створювати складні додатки та реагувати на події в реальному часі.

1.2 Детальний огляд веб-потоків та їх використання для обробки MQTT повідомлень

[6] Веб-потоки (Websockets) - це протокол комунікації, який дозволяє встановлювати двостороннє з'єднання між клієнтом та сервером через веб-браузер.

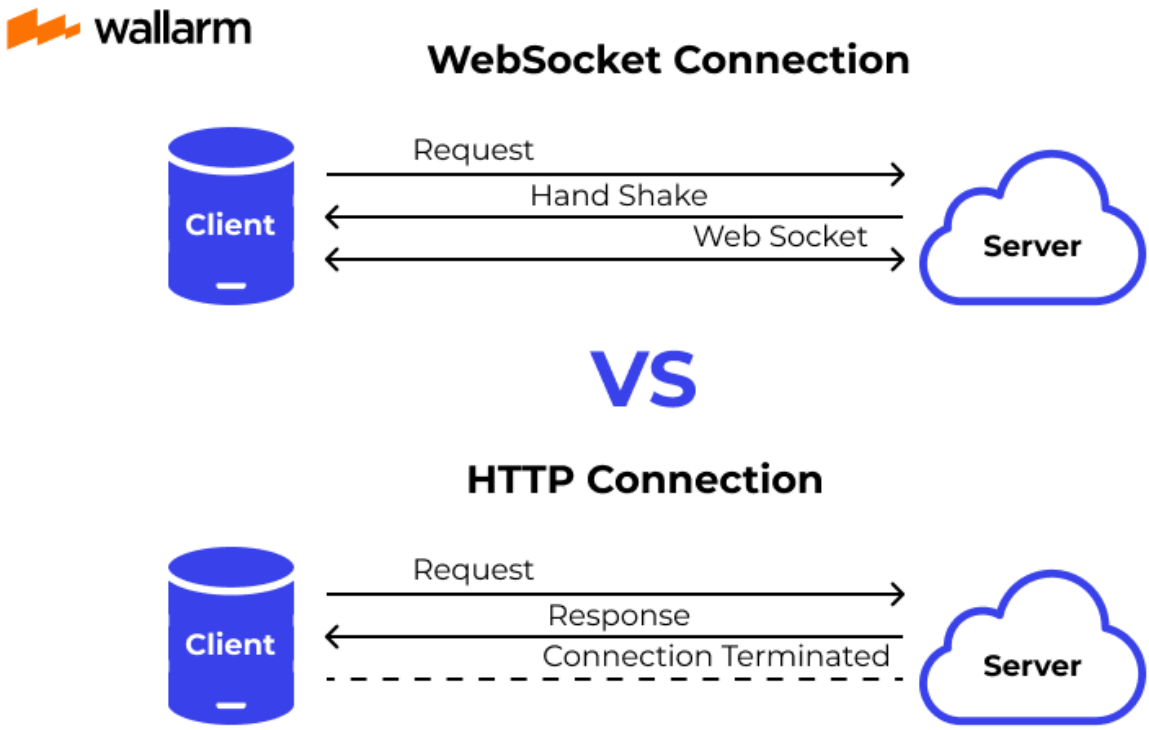


Рисунок 1.4 Принципи роботи Websocket та HTTP

Відповідно до загальноприйнятого визначення, WebSocket — це дуплексний протокол, який використовується переважно в каналі зв'язку клієнт-сервер. Це двонаправлений характер, що означає, що зв'язок відбувається туди-сюди між клієнтом і сервером.

З'єднання, розроблене за допомогою WebSocket, триває до тих пір, поки будь-яка зі сторін-учасниць не розриває його. Коли одна сторона розриває з'єднання, друга сторона не зможе спілкуватися, оскільки з'єднання розривається автоматично на передній частині.

Щоб ініціювати з'єднання, WebSocket потребує підтримки HTTP . Говорячи про його корисність, він є хребтом для розробки сучасних веб-додатків, коли йдеться про безперебійну передачу даних і різноманітний несинхронізований трафік.

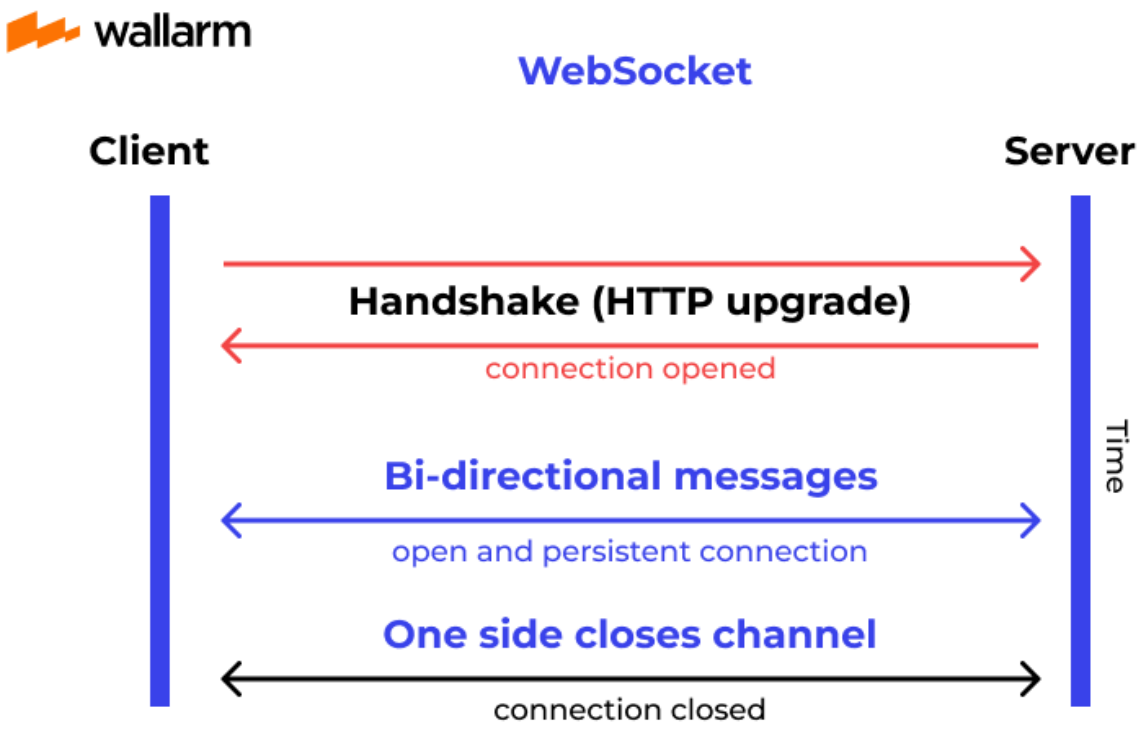


Рисунок 1.5 Принцип роботи Websocket

Цей протокол став важливим інструментом для реалізації веб-додатків в реальному часі, включаючи обробку MQTT повідомлень. Ось детальний огляд веб-потоків та їх використання для обробки MQTT повідомлень:

1. Загальна ідея веб-потоків:

Веб-потоки дозволяють клієнтам та серверам взаємодіяти між собою в реальному часі через стійке з'єднання.

Вони базуються на стандарті WebSocket, який визначає двосторонню комунікацію між клієнтом і сервером, яка залишається відкритою протягом всього сеансу.

2. Використання веб-потоків для обробки MQTT повідомлень:

MQTT є протоколом передачі повідомлень через мережу. Використання веб-потоків разом з MQTT дозволяє використовувати браузер для отримання та відправки MQTT повідомлень.

3. З'єднання з брокером MQTT:

Для використання веб-потоків з MQTT, клієнт повинен встановити з'єднання з брокером MQTT за допомогою WebSocket URL. Наприклад: ws://mqtt-broker.com:1883.

В браузері це може бути зроблено за допомогою бібліотеки, яка підтримує веб потоки для MQTT, такої як Paho JavaScript бібліотека.

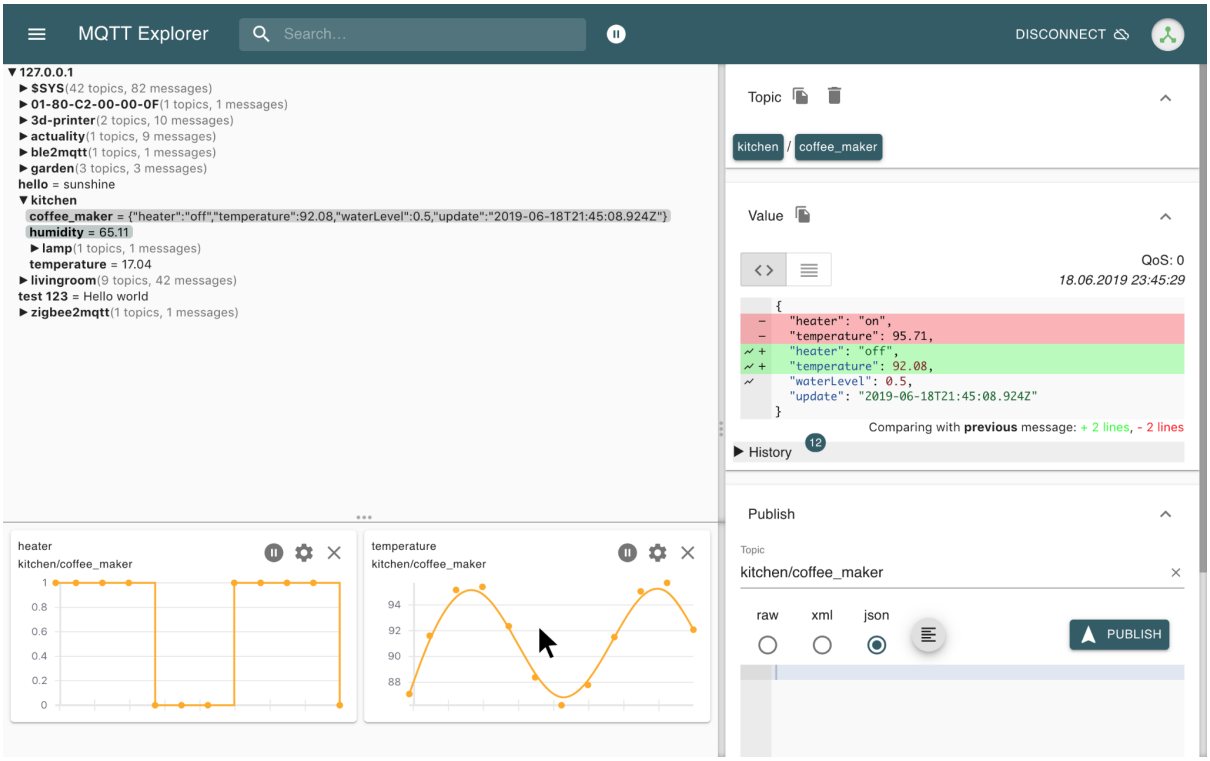


Рисунок 1.6 Застосунок MQTT Explorer

4. Підписка та публікація MQTT повідомлень:

Після встановлення з'єднання з брокером MQTT через веб-потоки, клієнт може підписатися на тематичні канали та отримувати MQTT повідомлення в реальному часі.

Також можливо відправляти MQTT повідомлення через це з'єднання.

5. Використання веб-потоків в веб-додатках:

Веб-потоки широко використовуються в веб-додатках для створення функцій реального часу, таких як чати, потоки новин, онлайн-ігри тощо.

Вони дозволяють негайно передавати дані між клієнтом та сервером без необхідності постійного опитування сервера.

6. Захист та безпека:

Важливо враховувати заходи безпеки при використанні веб-потоків, так як вони відкривають новий канал комунікації. Включення шифрування SSL/TLS рекомендується для забезпечення безпеки даних.

7. Переваги використання веб-потоків для MQTT:

Зменшення затримок у передачі даних.

Можливість відправляти дані до браузера та від нього, що відкриває нові можливості для веб-додатків.

Дозвіл робити веб-клієнти для MQTT без використання спеціалізованих бібліотек.

Використання веб-потоків для обробки MQTT повідомлень робить вас здатними реалізувати реально-часні веб-додатки, які спроможні обмінюватися даними з брокером MQTT в реальному часі через веб-браузери клієнтів. Це важливо в контексті розробки додатків IoT та M2M, де важлива швидкість та надійність обміну даними.

2 ВИКОРИСТАННЯ DOCKER ТА КОНТЕЙНЕРІВ ДЛЯ ОБРОБКИ IOT ДАНИХ

[7] InfluxDB - це високопродуктивна та розширювана система керування часовими рядами (Time Series Database - TSDB). Вона спеціально розроблена для зберігання, опрацювання та запитування даних, які мають часову компоненту, такі як дані сенсорів, метеодані, логи, метрики продуктивності, а також дані, що генеруються в інтернеті речей (IoT) і системах моніторингу.

Ось кілька ключових аспектів та понять, пов'язаних з InfluxDB:

1. Часові ряди (Time Series): В основі InfluxDB лежить ідея зберігання даних у вигляді часових рядів, де кожен запис має часовий мітка (timestamp). Це дозволяє виконувати операції з даними, що стосуються часу, з високою продуктивністю.

2. Модель даних: Дані в InfluxDB організовані в бази даних, в яких містяться рядки (measurements), кожен рядок містить часові ряди (time series). Часовий ряд складається з польотів (fields), тегів (tags) та часових міток (timestamp).

Fields: Це числові значення або дані плаваючої точки, які зберігаються в часовому ряді.

Tags: Це мітки або ключові слова, які допомагають індексувати дані та швидше виконувати запити.

Timestamp: Кожен запис в часовому ряді має часову мітку, що визначає, коли були збережені дані.

3. Мова запитів: InfluxDB використовує свою власну мову запитів, відому як InfluxQL, а також підтримує запити через REST API та SQL-подібний запит мови, що полегшує виконання складних запитів до часових рядів.

4. Масштабованість: InfluxDB розроблено з урахуванням масштабованості. Ви можете розгортати InfluxDB в режимі хмари або власному центрі обробки даних та розширювати його за потребою.

5. Візуалізація даних: InfluxDB може легко інтегруватися з інструментами візуалізації, такими як Grafana, щоб відобразити дані у вигляді графіків та діаграм.

6. Зберігання великих обсягів даних: InfluxDB може зберігати великі обсяги даних, навіть у видачі високої швидкості запису.

7. Висока продуктивність: Інструменти InfluxDB розроблені для виконання операцій з даними у великій кількості та у реальному часі, що дозволяє вам миттєво аналізувати та реагувати на дані.

InfluxDB є потужним інструментом для зберігання та аналізу даних з часовою компонентою, особливо в сферах IoT, моніторингу, аналітики логів та багатьох інших областях, де важлива обробка та аналіз часових даних.

Docker для IoT

[8] Docker — це контейнерна відкрита платформа для розробки, розгортання та запуску програм. Контейнерізація дає змогу створити контейнер, який містить вашу програму, будь-які двійкові файли чи бібліотеки, від яких залежить ваша програма, і будь-які деталі конфігурації. Потім ви можете запускати програмне забезпечення на кількох машинах без великої кількості налаштувань.

Докери називають це контейнерами, тому що ви можете уявити це як транспортний контейнер. Транспортні контейнери мають стандартний розмір, тому їх можна переміщувати в портах і доставляти морем або сушею. Вони також можуть містити майже все. Контейнери Docker можуть містити код вашого програмного забезпечення та його залежності, щоб його можна було легко запускати на багатьох різних машинах. Розробники часто використовують їх для створення сервера веб-додатків, який працює на їхній власній машині для розробки, а потім надсилається в хмару для загального використання.

Спосіб роботи контейнерних програм показано нижче.

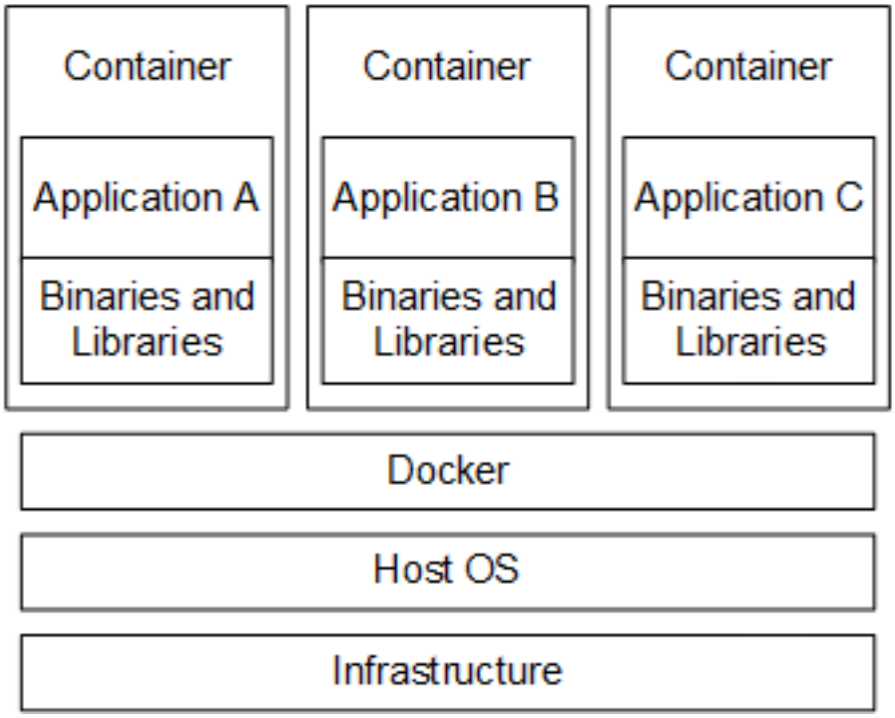


Рисунок 2.1 Спосіб роботи контейнерних програм

Спосіб роботи контейнерних програм

Як видно з цієї діаграми, контейнер Docker містить програму, а також будь-які двійкові файли або бібліотеки, які потрібні програмі для її роботи. Контейнер працює під керуванням Docker, який, у свою чергу, працює поверх операційної системи.

Переваги використання Docker

Запуск програм у контейнерах дає низку переваг:

• Портативність

Після того, як ви протестували свою контейнерну програму, ви можете розгорнути її в будь-якій іншій системі, де працює Docker, і ви можете бути впевнені, що ваша програма працюватиме так само, як і під час тестування.

• Продуктивність

Незважаючи на те, що віртуальні машини є альтернативою контейнерам, той факт, що контейнери не містять операційної системи (тоді як віртуальні машини містять), означає, що контейнери мають набагато менші розміри, ніж віртуальні машини, швидше створюються та швидше запускаються.

• Спритність

Переваги портативності та продуктивності, які пропонують контейнери, можуть допомогти вам зробити процес розробки більш гнучким і оперативним. Удосконалення процесів безперервної інтеграції та безперервної доставки, щоб скористатися перевагами контейнерів і технологій, таких як Enterprise Developer Build Tools для Windows, спрощує доставку потрібного програмного забезпечення в потрібний час. Enterprise Developer Build Tools для Windows — це компонент Enterprise Developer, який надає всі функції Enterprise Developer, щоб ви могли компілювати, створювати та тестувати код COBOL, але без додаткових витрат на IDE.

• Ізоляція

Контейнер Docker, який містить одну з ваших програм, також містить відповідні версії будь-якого допоміжного програмного забезпечення, яке вимагає ваша програма. Якщо інші контейнери Docker містять програми, які вимагають різних версій того самого допоміжного програмного забезпечення, це не проблема, оскільки різні контейнери Docker повністю незалежні один від одного.

Це також означає, що під час проходження різних етапів життєвого циклу розробки ви можете бути повністю впевнені, що створене вами під час розробки зображення працюватиме точно так само, як воно проходитиме через тестування та потенційно для ваших користувачів.

• Масштабованість

Ви можете швидко створювати нові контейнери, якщо цього потребують ваші програми. Використовуючи кілька контейнерів, ви можете скористатися низкою параметрів керування контейнерами. Перегляньте документацію Docker, щоб дізнатися більше про ці параметри.

Чому IoT використовує Docker (конкретизація переваг у використанні IoT)

Контейнери добре підходять для вирішення деяких основних проблем, з якими стикаються розробники під час розгортання програмного забезпечення на пристроях IoT:

▪ Мінімальні апаратні ресурси. Багатьом пристроям IoT не вистачає потужних обчислювальних ресурсів і пам’яті. Тому їх здатність обробляти оновлення

програмного забезпечення обмежена. Контейнери можуть допомогти в цьому, оскільки встановлення нового образу контейнера не потребує великої обчислювальної потужності. Для пристрою IoT потрібно просто завантажити зображення, розмістити його там, де воно буде жити, і видалити старе зображення. Обробка конфігурації мінімальна.

▪ Географічний розподіл. У деяких випадках використання пристроїв IoT розкидано по великій географічній території. Доставка програмного забезпечення для них з єдиного центрального сховища може не працювати належним чином. За допомогою Docker легко створювати реєстри зображень у кількох місцях, щоб добре обслуговувати всю мережу.

▪ Обмежений або спорадичний доступ до мережі. Незважаючи на значення терміна «Інтернет речей», не всі пристрої в Інтернеті речей добре підключені до Інтернету. Вони можуть мати обмежену пропускну здатність мережі або бути онлайн лише час від часу. Docker може допомогти доставити оновлення програмного забезпечення за таких обставин, оскільки під час оновлення зображень контейнерів Docker завантажує лише ті частини зображення, які змінилися.

▪ Різноманітні середовища пристрою. Програмне забезпечення, яке працює на пристрої IoT, може бути майже будь-яким. Різноманітність конфігурацій програмного забезпечення на пристроях IoT зазвичай ускладнює встановлення додатків, оскільки додатки повинні бути налаштовані для кожного типу середовища, якщо вони встановлені традиційними методами. Однак для контейнерів версія операційної системи та інші програмні змінні набагато менш важливі. Якщо пристрій працює під керуванням якогось дистрибутива Linux і має середовище виконання контейнера, ви можете встановити на ньому контейнерну програму без спеціального налаштування.

Опис схеми даних для зберігання MQTT даних

[9] Схема даних для зберігання MQTT даних в InfluxDB може бути досить простою, і вона включає в себе наступні ключові елементи:

Бази даних (Databases): Почнемо з створення бази даних для зберігання MQTT даних. Кожна база даних може відповідати певному проекту, пристрою або виміру. Наприклад, "IoT_Sensors" або "Home_Automation".

Часові ряди (Time Series): Всередині кожної бази даних ви створюєте часові ряди, які відображають окремі сенсори або параметри, які ви бажаєте відстежувати в часі. Кожен часовий ряд має унікальну назву та структуру даних.

Fields - це числові значення або дані плаваючої точки, які відображають фактичні виміри з MQTT повідомлень. Наприклад, для датчика температури значення температури може бути field.

Tags - це мітки або ключові слова, які допомагають організовувати дані та надавати їм контекст. Наприклад, для датчика температури ви можете мати теги, які вказують на його розташування або тип.

Timestamp: Кожен запис в часовому ряді має часову мітку, що вказує, коли були отримані дані від MQTT повідомлення. Timestamp дозволяє вам проводити запити за певний період часу.

Просторові виміри (Measurement): Просторові виміри можуть використовуватися для групування схожих часових рядів в межах бази даних. Наприклад, у базі даних "IoT_Sensors" можуть бути просторові виміри "Temperature", "Humidity", "Pressure" і т.д.

Наприклад, ось приклад схеми даних для зберігання температурних даних в InfluxDB:

- База даних: "IoT_Sensors"
- Просторовий вимір: "Temperature"
- Часовий ряд: "sensor_1_temperature"
    - Fields:
        - "value": значення температури
    - Tags:
        - "location": розташування датчика
        - "sensor_type": тип датчика

Така структура дозволяє організовувати та зберігати дані з MQTT повідомлень у вигляді часових рядів з різними параметрами і тегами для подальшого аналізу та візуалізації.

Розгляд SQL-подібної мови запитів InfluxQL

[10] InfluxQL - це мова запитів, яка використовується для взаємодії з базами даних InfluxDB. Вона є SQL-подібною мовою та призначена для запитування, агрегування, фільтрації та аналізу даних в часових рядах. Ось кілька основних концепцій та приклади запитів InfluxQL:

1. Вибірка (SELECT): Основна команда для вибору даних з бази даних InfluxDB.

sql

SELECT * FROM measurement_name WHERE tag_name = 'tag_value' AND time > '2023-01-01T00:00:00Z'

- SELECT *: Вибір всіх полів з вказаного виміру.
- FROM measurement_name: Вказує вимір, з якого ви хочете вибрати дані.
- WHERE tag_name = 'tag_value' AND time > '2023-01-01T00:00:00Z': Умова для фільтрації результатів за значеннями тегів та часовою міткою.
- 2. Агрегація (GROUP BY): Дозволяє агрегувати дані в різних інтервалах часу.

  sql

  SELECT MEAN("value") FROM measurement_name WHERE time > now() - 1h GROUP BY time(10s), "tag_name"

- MEAN("value"): Обчислення середнього значення поля "value".
- GROUP BY time(10s), "tag_name": Агрегування за інтервалами 10 секунд та за значеннями тега "tag_name".

  3. Запис даних (INSERT): Дозволяє вставляти дані у базу даних.

  sql

  INSERT measurement_name,tag_name=tag_value field_name=field_value

- INSERT measurement_name: Вказує вимір, в який буде вставлена нова запис.
- tag_name=tag_value: Вказує значення тегу.
- field_name=field_value: Вказує значення поля.

4. Видалення даних (DELETE): Дозволяє видаляти дані з бази даних.

sql

DELETE FROM measurement_name WHERE time < '2023-01-01T00:00:00Z'

- DELETE FROM measurement_name: Вказує вимір, з якого будуть видалені дані.
- WHERE time < '2023-01-01T00:00:00Z': Умова для видалення записів, що відповідають певному часовому діапазону.

5. Перегляд бази даних (SHOW): Дозволяє переглядати список доступних баз даних, вимірів, ретенцій та інших об'єктів.

sql

SHOW DATABASES SHOW MEASUREMENTS SHOW RETENTION POLICIES ON database_name

Це лише кілька основних прикладів запитів InfluxQL. Мова має багато інших функцій, таких як агрегація за функціями (SUM, COUNT, MIN, MAX і т.д.), визначення власних функцій, робота з групами та інші можливості, які дозволяють ефективно аналізувати та опрацьовувати дані в InfluxDB.

Огляд можливостей візуалізації даних в Grafana

[11] Grafana - це потужний інструмент для візуалізації даних та моніторингу, який дозволяє створювати інтерактивні та інформативні графіки, панелі та дахборди для аналізу та відображення даних з різних джерел. Ось огляд можливостей візуалізації даних в Grafana:

1. Підтримка різних джерел даних: Grafana підтримує багато різних джерел даних, таких як InfluxDB, Prometheus, Elasticsearch, MySQL, PostgreSQL, Microsoft SQL Server, AWS CloudWatch, та багато інших. Ви можете інтегрувати свої дані з різних джерел для подальшої візуалізації та аналізу.

2. Широкий вибір графіків та візуальних елементів: Grafana надає великий набір графіків, діаграм, таблиць, стовпчатих діаграм, географічних карт та інших візуальних елементів, які можна використовувати для створення різноманітних видів візуалізацій.

3. Інтерактивність та взаємодія: Ви можете створювати інтерактивні графіки та панелі, які дозволяють користувачам взаємодіяти з даними. Наприклад, ви можете дозволити їм фільтрувати дані за певними параметрами або вибирати відображені дані на графіку.

4. Спеціалізовані плагіни та розширення: Grafana має велику кількість спеціалізованих плагінів та розширень, які дозволяють вам створювати специфічні типи візуалізацій, такі як графіки для моніторингу продуктивності, теплові карти, схеми будівель, графіки для моніторингу IoT даних і багато інших.

5. Реалізація графіків та панелей для моніторингу: Grafana добре підходить для створення графіків та панелей для моніторингу систем та мережі. Ви можете легко створювати панелі моніторингу для відстеження стану ресурсів, використовуючи дані з різних джерел.

6. Підтримка сповіщень: Grafana дозволяє налаштовувати сповіщення на основі ваших візуалізацій, щоб отримувати повідомлення про події або проблеми, коли вони виникають.

7. Розкладання графіків на кілька рядків та сторінок: Ви можете організовувати графіки та панелі у вигляді рядків та сторінок, щоб створювати комплексні дахборди для відображення великої кількості інформації.

8. Гнучка настройка: Grafana надає широкі можливості налаштування стилю та вигляду графіків, шрифтів, кольорів та інших елементів візуалізації.

9. Групування та додавання анотацій: Можливість групувати графіки, додавати анотації та коментарі допомагає в створенні більш зрозумілих та інформативних візуалізацій.

10. Підтримка ролей та прав доступу: Grafana дозволяє налаштовувати рівні доступу та контролювати, хто може переглядати та редагувати візуалізації.

Grafana є дуже потужним інструментом для візуалізації та моніторингу даних, що робить його популярним у різних областях, включаючи моніторинг систем, IoT, аналітику даних та багато інших. З його допомогою можна легко створювати зрозумілі та вражаючі візуалізації для аналізу ваших даних.

Створення і налаштування графіків та панелей для моніторингу MQTT даних

[12] Створення і налаштування графіків та панелей для моніторингу MQTT даних в Grafana включає кілька кроків. Ось загальний підхід:

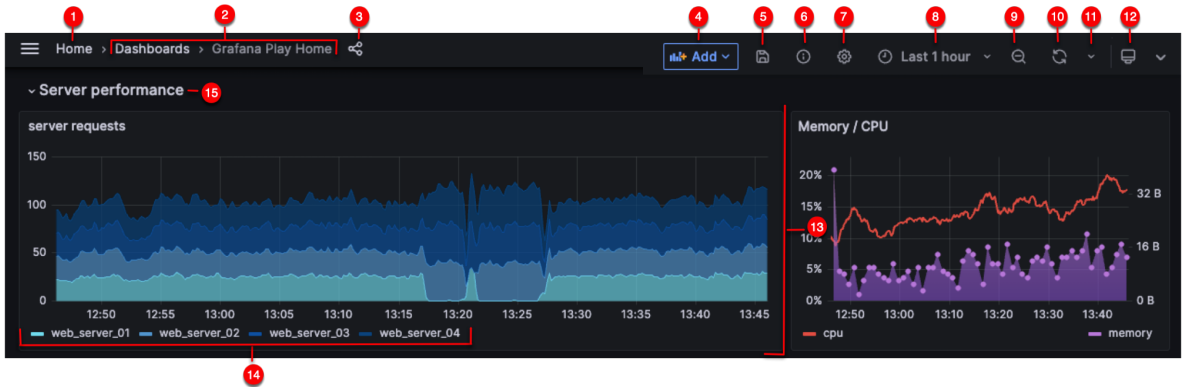


Рисунок 2.2 Створення і налаштування графіків та панелей для моніторингу MQTT даних в Grafana

1. Домашня сторінка Grafana: клацніть «Домашня сторінка» в навігаційній доріжці, щоб перейти на домашню сторінку, налаштовану в екземплярі Grafana.

2. Заголовок інформаційної панелі: клацнувши назву інформаційної панелі, ви можете шукати інформаційні панелі, які містяться в поточній папці.

3. Поділитися інформаційною дошкою або панеллю: скористайтеся цією опцією, щоб поділитися поточною інформаційною дошкою або панеллю за допомогою посилання або знімка. Ви також можете експортувати визначення інформаційної панелі з модального доступу.

4. Додати: використовуйте цей параметр, щоб додати панель, рядок панелі інструментів або панель бібліотеки до поточної панелі інструментів.

5. Зберегти інформаційну панель: натисніть, щоб зберегти зміни на інформаційній панелі.

6. Статистика інформаційної панелі: клацніть, щоб переглянути аналітику інформаційної панелі, включаючи інформацію про користувачів, активність, кількість запитів. Дізнайтеся більше про аналітику інформаційної панелі.

7. Налаштування інформаційної панелі: використовуйте цей параметр, щоб змінити ім’я інформаційної панелі, папку та теги, а також керувати змінними та запитами на анотації. Докладніше про налаштування інформаційної панелі.

8. Розкривне меню вибору часу: клацніть, щоб вибрати параметри відносного діапазону часу та встановити спеціальні абсолютні діапазони часу.

Ви можете змінити параметри часового поясу та фінансового року в елементах керування часовим діапазоном, натиснувши кнопку Змінити налаштування часу.

Параметри часу зберігаються на кожній інформаційній панелі.

9. Зменшити діапазон часу: натисніть, щоб зменшити діапазон часу. Дізнайтеся більше про те, як використовувати загальні елементи керування діапазоном часу.

10. Оновити інформаційну панель: натисніть, щоб негайно запустити запити та оновити дані інформаційної панелі.

11. Інтервал часу оновлення інформаційної панелі: натисніть, щоб вибрати інтервал часу автоматичного оновлення інформаційної панелі.

12. Режим перегляду: клацніть, щоб відобразити інформаційну панель на великому екрані, наприклад телевізорі чи кіоску. Режим перегляду приховує нерелевантну інформацію, наприклад навігаційні меню. Дізнайтеся більше про режим перегляду в нашому блозі «Як створити кіоски для відображення інформаційних панелей на телевізорі».

13. Панель приладів: основним блоком приладів є панель. Щоб додати нову панель, рядок панелі інструментів або панель бібліотеки, натисніть «Додати панель».

Панелі бібліотеки можна спільно використовувати багатьма інформаційними панелями.

Щоб перемістити панель, перетягніть заголовок панелі в інше місце.

Щоб змінити розмір панелі, натисніть і перетягніть нижній правий кут панелі.

14. Легенда графіка: змінюйте кольори серії, вісь ординат і видимість серії безпосередньо з легенди.

15. Рядок інформаційної панелі: рядок інформаційної панелі — це логічний роздільник на інформаційній панелі, який групує панелі разом.

Рядки можна згортати або розгортати, що дозволяє приховати частини інформаційної панелі.

Панелі всередині згорнутого рядка не видають запити.

Використовуйте повторювані рядки для динамічного створення рядків на основі змінної шаблону.

Тепер ви можете створити та налаштувати графіки та панелі для відображення MQTT даних в Grafana. Графіки будуть оновлюватися в реальному часі, якщо ваші MQTT дані змінюються, і ви зможете легко відстежувати стан системи або пристроїв.

3 РОЗГОРТАННЯ СИСТЕМИ ОБРОБКИ ТА МОНІТОРИНГУ MQTT ДАНИХ

Прототип обладнання

[13] Для всіх доказів концепції, які приведені в цьому проекті, ми проведемо віртуальне моделювання, але ми також впровадимо реальні датчики. Для цього ми підключимо наш Raspberry Pi 3 B+ до пристроїв ESP32/ESP8266 через MQTT. До одного з ESP32 ви підключите пристрій температури, вологості, тиску та висоти, наприклад BMP280. В іншому буде підключено світлодіод, який засвітиться, коли активується попередньо запрограмоване сповіщення.

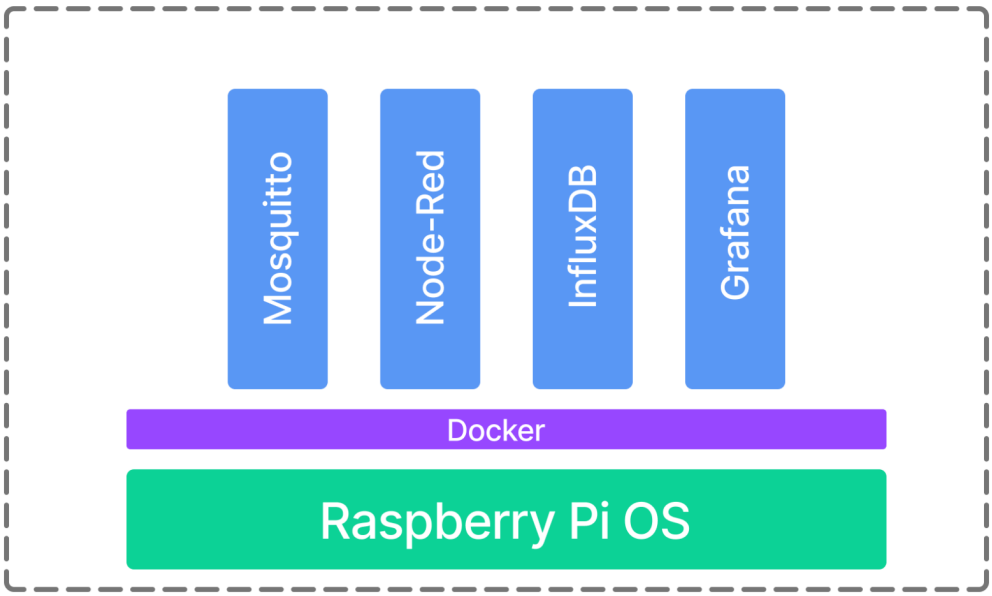


Рисунок 3.1 Архітектура обробки даних Mosquito, Node-red, Influxdb, Grafana

Архітектура обробки даних Mosquito Node-red Influxdb Grafana

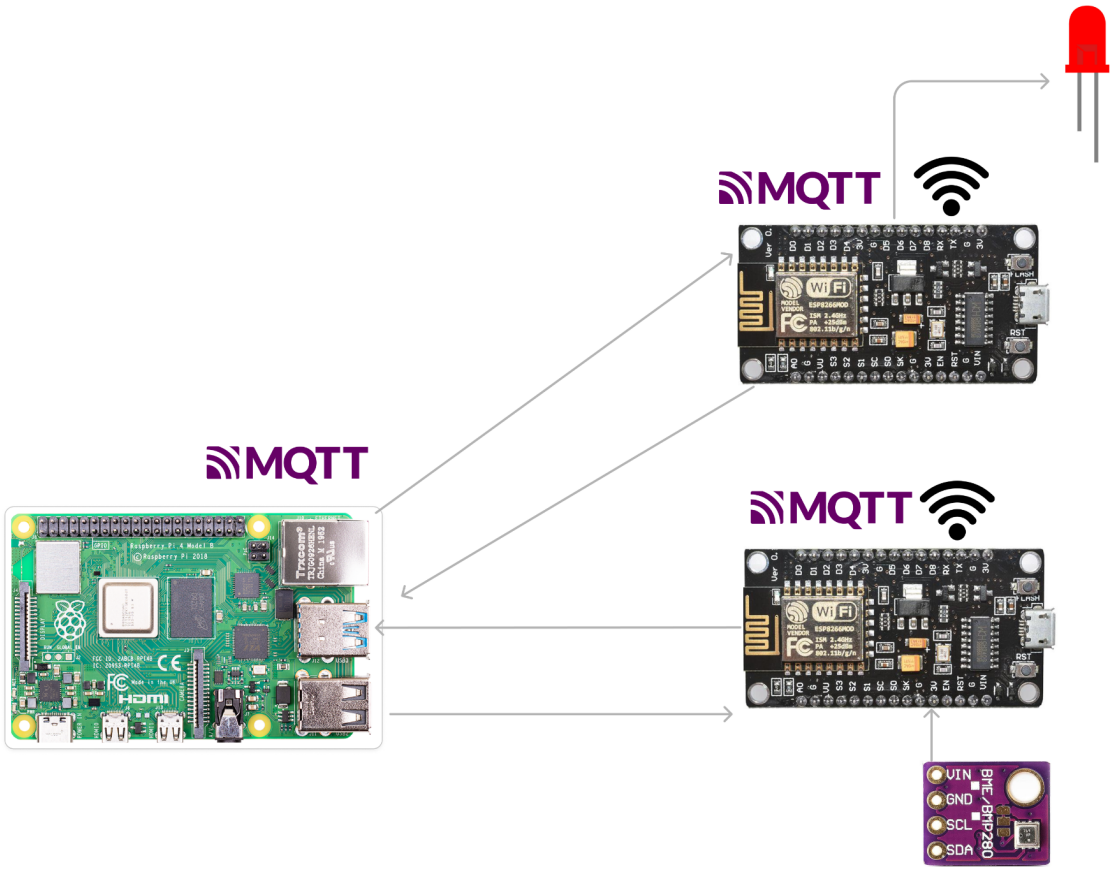


Рисунок 3.2 Архітектура системи: Raspberry Pi 3B+, ESP8266 з сенсором BME280 та ESP8266 з світлодіодом для сповіщень

Архітектура системи: Raspberry Pi 3B+, ESP8266 з сенсором BME280 та ESP8266 з світлодіодом для сповіщень.

Розробка архітектури системи обробки та моніторингу MQTT повідомлень

[14] Розробка архітектури системи обробки та моніторингу MQTT повідомлень на основі Node-Red, InfluxDB і Grafana може бути розділена на кілька ключових компонентів. Ось загальний огляд архітектури:

MQTT Broker: Центральним елементом системи є MQTT брокер. Він відповідає за приймання, збереження та розповсюдження MQTT повідомлень. Брокер забезпечує комунікацію між MQTT-клієнтами та іншими частинами системи. [28]

Node-Red використовується для обробки та маршрутизації MQTT повідомлень. Ви можете налаштовувати потоки даних, виконувати логіку,

фільтрацію та трансформацію даних, а також відправляти дані до InfluxDB для зберігання.

InfluxDB служить як часова база даних для зберігання MQTT даних. Ви можете створити бази даних для різних типів даних та вимірів, а також налаштовувати ретенцію для збереження старих даних.

Grafana використовується для візуалізації та моніторингу MQTT даних. Ви можете створювати панелі та графіки для відображення даних з InfluxDB та налаштовувати сповіщення для слідкування за станом системи.

MQTT-клієнти відправляють та отримують MQTT повідомлення в системі. Вони можуть бути підключені до датчиків, пристроїв або інших джерел даних, які відправляють дані до MQTT брокера. [29]

Веб-додатки та інші інтеграції: Ви також можете інтегрувати систему з іншими веб-додатками, службами або іншими системами для розширення функціональності та обробки MQTT даних.

Архітектура системи обробки та моніторингу MQTT повідомлень повинна бути добре розділеною на компоненти, які співпрацюють між собою, і мати механізми керування та моніторингу. Важливо також враховувати швидкість обміну даними, надійність та безпеку для систем IoT та моніторингу в реальному часі.

3.1 Встановлення та налаштування Docker контейнера

[15] Для встановлення Docker можна використовувати будь який компютер з встановленою ОС Linux. У цьому досліджені я використовував одноплатнй компютер Raspberry Pi 3B+ з Rasspberry OS Desctop

Завантажте та виконайте скрипт за допомогою команди:

curl -sSL https://get.docker.com | sh

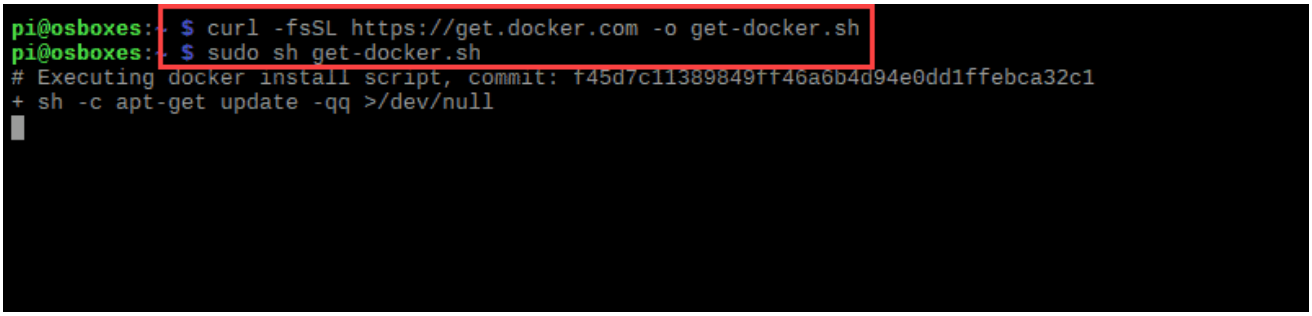


Рисунок 3.3 Запуск скрипта завантаження Docker

Це встановить необхідні пакунки для вашого дистрибутива Raspbian Linux.

Результат покаже вам, яка версія Docker зараз запущена у вашій системі. [30]

Налаштування Docker

[16] Щоб отримати найкращий досвід, необхідно виконати кілька кроків вручну.

Налаштуйте Docker на автоматичний запуск:

sudo systemctl enable docker

Тепер ви можете перезавантажити Pi (за допомогою команди sudo shutdown -r now) або запустити

Запуск демона docker:

sudo systemctl run docker

Увімкнути клієнт Docker:

Клієнт Docker може використовуватися лише адміністратором або членами групи докерів. Додайте користувача pi або еквівалентного користувача до групи докерів:

sudo usermod -aG docker pi

Якщо команда успішно виконана, в терміналі нічого не відбудеться. Щоб зміни відбулися, вам потрібно вийти з сеансу користувача, а потім знову ввійти. Це дуже важливо для отримання правильних дозволів.

Перевірте версію Docker на Raspberry Pi, ввівши:

docker version

Вихідні дані відображатимуть версію Docker разом із додатковою інформацією.

Щоб отримати загальносистемну інформацію (включаючи версію ядра, кількість контейнерів і зображень, а також більш розширений опис), виконайте:

docker info

За допомогою наступної команди ми побачимо контейнери, запущені в цьому контейнері докерів (на даний момент список контейнерів порожній): [31]

docker ps

Оновлення Docker на Raspberry Pi

Немає необхідності повторно запускати скрипт, щоб оновити Docker. Крім того, повторний запуск скрипту може спричинити проблеми, якщо він спробує відкрити вже додані сховища.

Оновіть Docker за допомогою менеджера apt пакетів командою:

sudo apt-get update

sudo apt-get upgrade

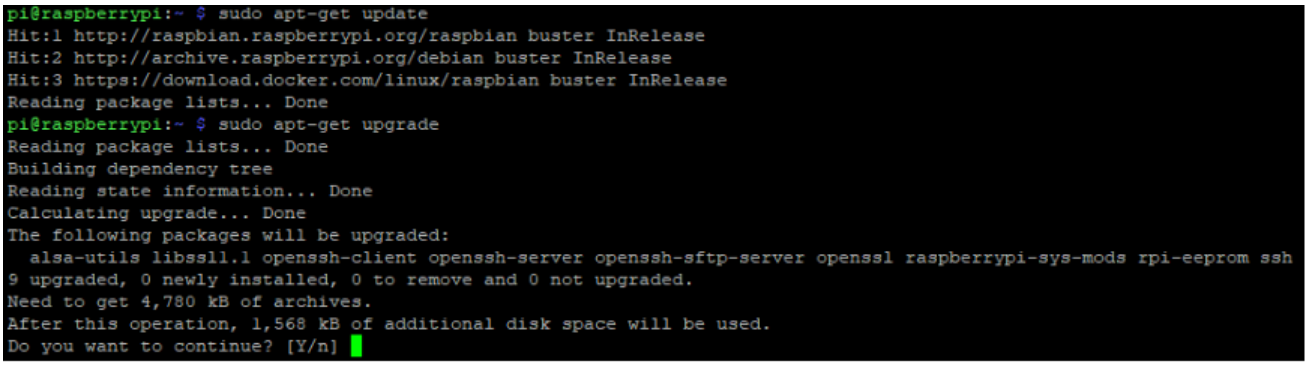


Рисунок 3.3 Оновлення пакетів linux

Встановлення застосунку MQTT Broker

[17] Спочатку завантажте останній образ контейнера mosquitto з Docker Hub:

docker pull eclipse-mosquitto

Створіть папки для MQTT

Каталоги, замість логічних томів створюються для додавання всіх даних контейнера та легшого доступу до них. Каталоги створюються в корені, щоб уникнути конфліктів дозволів [32]

sudo su

cd /

mkdir docker/mosquitto

mkdir docker/mosquitto/config

mkdir docker/mosquitto/data

mkdir docker/mosquitto/log

Для створення початкового конфігураційного файлу необхідно створити текстовий файл під назвою mosquitto.conf у щойно створеному підкаталозі mosquitto/config/ і відредагувати його:

```
nano /docker/mosquitto/config/mosquitto.conf
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log
#password_file /mosquitto/config/pwfile
allow_anonymous true
listener 1883 192.168.0.10
```

Рисунок 3.5 Конфігурація MQTT брокера

Команда для запуску контейнера mosquitto досить довга, оскільки потрібно вказати каталог, де зберігати файли. Якщо контейнер буде зупинено, або перестворено файли залишаться:

sudo docker run --name mosquitto -itd --restart=unless-stopped -- net=host -v /docker/mosquitto/config:/mosquitto/config -v /docker/mosquitto/data:/mosquitto/data -v / docker/mosquitto/log:/mosquitto/log -v /docker/mosquitto/config/mosquitto.conf:/mosquitto/config/mosquitt o.conf eclipse-mosquitto

--name: назва контейнера

-i: інтерактивний режим термінала

-t: дозволити TTY

-d: зберігати контейнер і поточну сесію терміналу окремо

--restart: докер не зупиниться, якщо машину вимкнено, лише якщо користувач зупинив роботу вручну

--net: підключити контейнер до мережі

-v: прив’язати монтувати том (host-source:container-dest)

[33] Примітка. Якщо з’являється повідомлення про помилку «Помилка відповіді від демона: Конфлікт. Назва контейнера» / mosquitto «уже використовується контейнером», необхідно видалити виявлений докер за допомогою команди sudo docker mosquitto rm -fv / mosquitto

Примітка. Якщо під час запуску docker ps ви помітили, що статус брокера не активний, перевірте, чи у вас уже є активний брокер mqtt (навіть якщо він не є докером), і вимкніть його або змініть порти в будь-якому з них.

[34] Перевірка працездатності

Відкрийте файл mosquitto.log з каталогу mosquitto/log і перевірте встановлену версію mosquitto: mosquitto -v

Для налаштування імені користувача та пароля необхідно увійти в інтерактивний термінал (sh - командна оболонка Shell) контейнера MQTT Для цього потрібно виконати команду:

docker exec -it mosquitto sh

Після потрібео виконати команду mosquitto_passwd з mosquitto bash, щоб додати користувача та його пароль (який буде збережено у форматі хешу):

cd mosquitto/config

mosquitto_passwd -c pwfile pi

Рисунок 3.6 Налаштування користувача та паролю mqtt

Встановлення Docker Node-RED

[18] Завантажте останній образ Nod-Red з Docker Hub:

docker pull nodered/node-red:latest

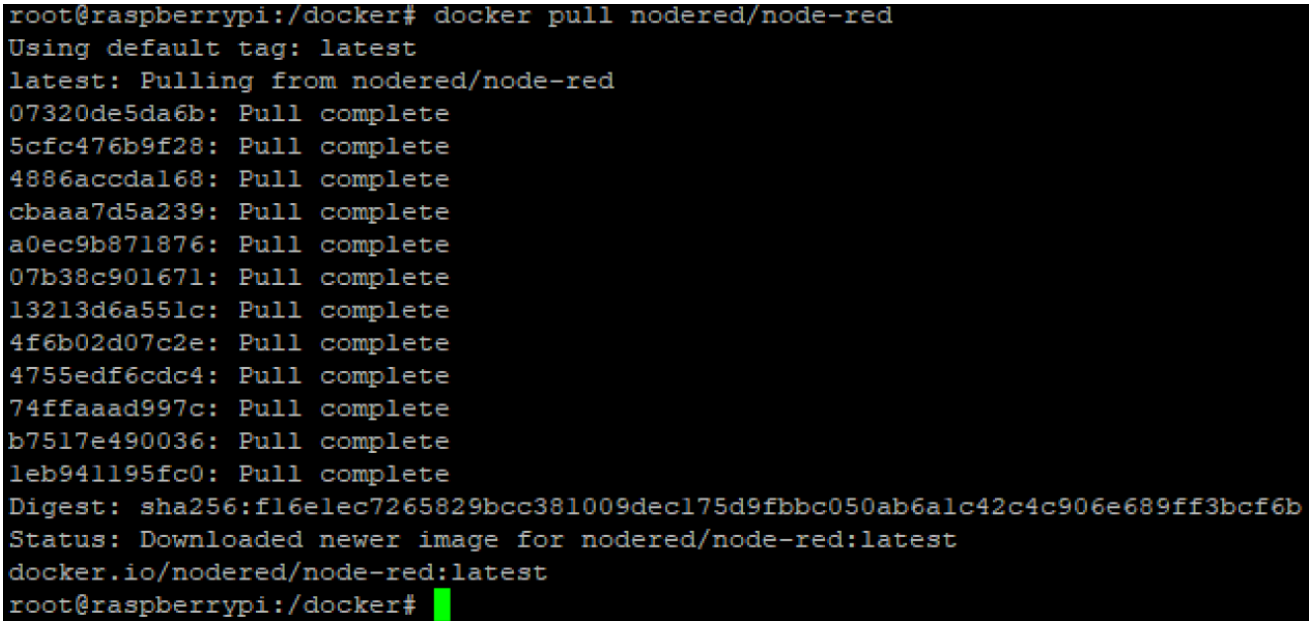


Рисунок 3.7 Процес встановлення контейнера Node-Red

Процес встановлення контейнера Node-Red подібний до попереднього MQTT. Щоб дані контейнера зберігалися навіть після перезапуску машини або докера. Для цього створимо каталог у папці «docker», і переконаємось, що будь-який існуючий каталог /data має правильне право власності. [35]

Після цього виконайте команду для запуску Node-RED Docker:

sudo su cd /

mkdir /docker/mynodered

mkdir /docker/mynodered/data

Нам потрібно буде переконатися, що будь-який існуючий каталог /data має правильне право власності. Це має бути 1000:1000. Це можна примусово виконати командою:

sudo chown -R 1000:1000 /docker/mynodered/data

Для запуску контейнера із правильною конфігурацією та каталогами виконайте команду:

sudo docker run --name mynodered -itd --restart=unless-stopped -p 1880:1880 -v /docker/mynodered/data:/data nodered/node-red

Після того в браузері можна перейти за адресою http://localhost:1880 та побачити інтерфейс [36]

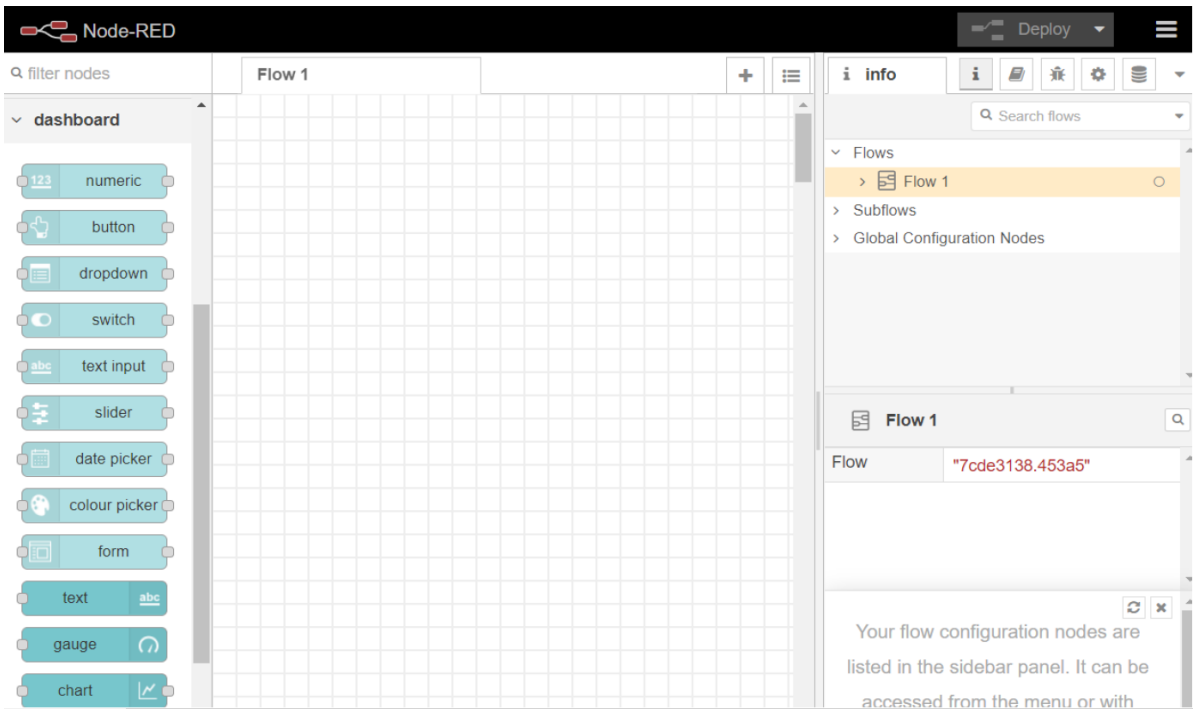


Рисунок 3.8 Встановлення панелі керування Node-RED

3.2 Налаштування вузлів Node-RED

[19] Модуль node-red-dashboard надає набір вузлів у Node-RED для швидкого створення інформаційної панелі живих даних.

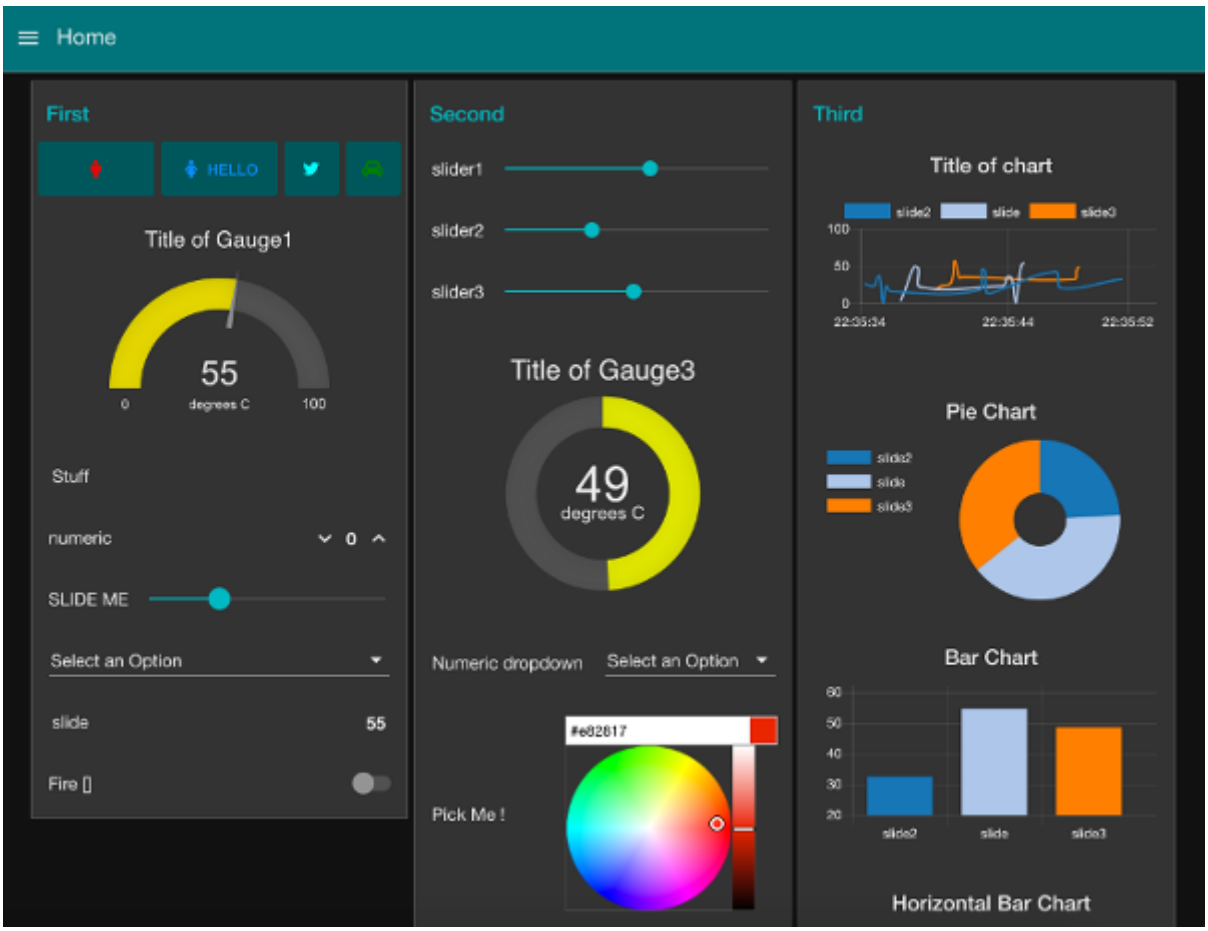


Рисунок 3.9 Інтерфейс вузла node-red-dashboard

[37] Щоб встановити вузол панелі через інтерфейс користувача Node-RED:

• Виберіть значок у верхньому правому куті

• Виберіть опцію «Manage pallete»:

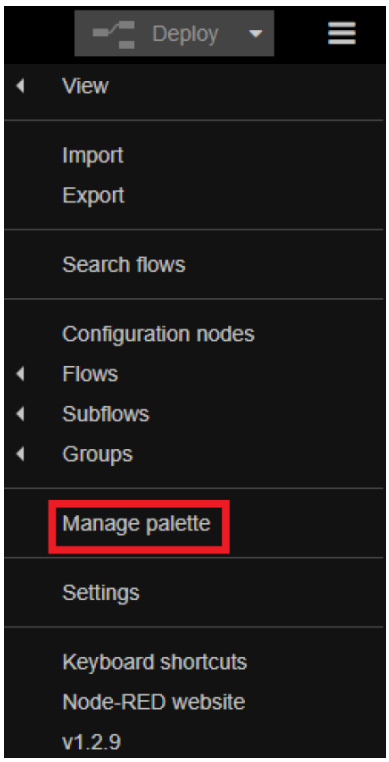


Рисунок 3.10 Панель керування Node-RED

Виберіть вкладку «Встановити», знайдіть node-red-dashboard і натисніть «встановити»:

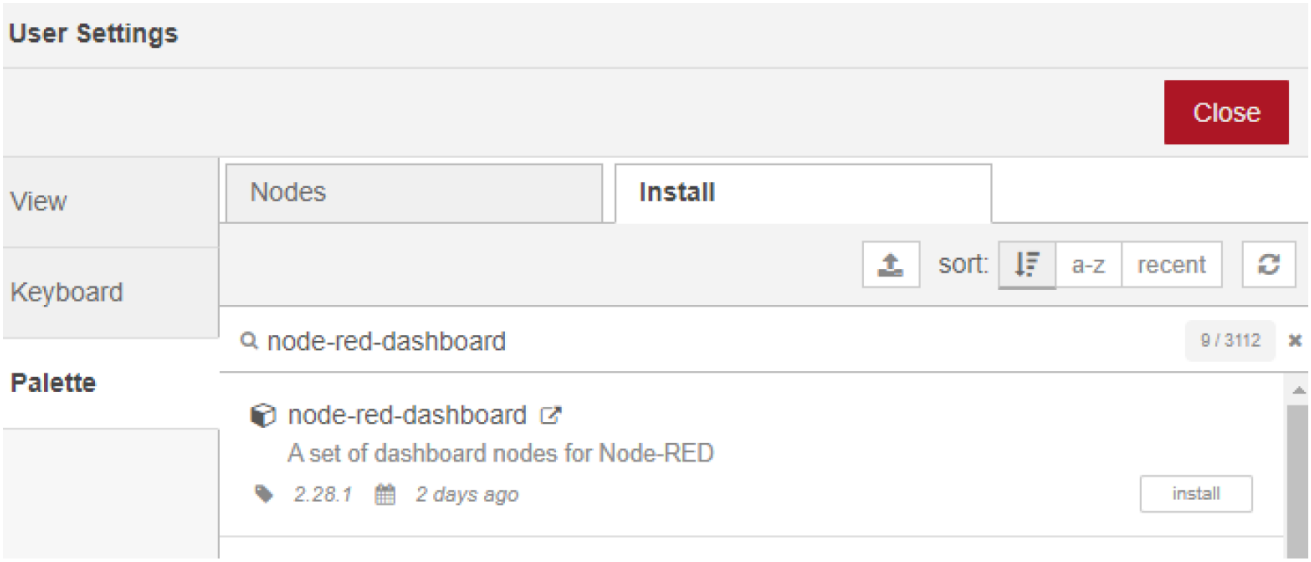


Рисунок 3.11 Пошук та встановлення вузлів Node-RED

Виберіть «Установити», щоб дозволити перезапуск Node-RED для встановлення модулів вузла [38]

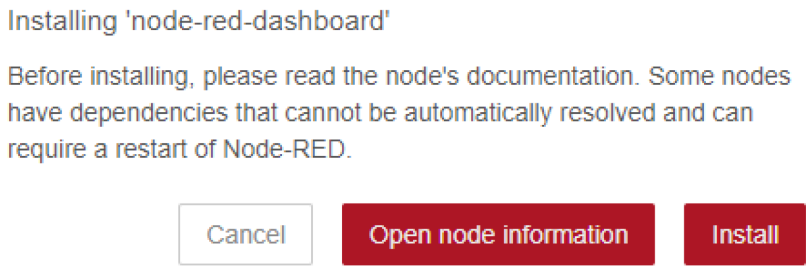


Рисунок 3.12 Встановлення MQTT брокера в Node-RED

Порівняння брокерів MQTT Mosquitto та Aedes [20] Mosquitto - легкий брокер з відкритим вихідним кодом, написаний мовою C. Ймовірно, найпопулярніший брокер MQTT. Підтримує MQTT v3.1.1 і v5

Aedes - дуже простий брокер, який ідеально підходить для розгортання невеликих домашніх мереж і вивчення MQTT, базується на Node.jsю У порівнянні з mosquitto він не надто багатий функціями, але ідеально підходить для проектів домашньої автоматизації. Підтримує MQTT v3.1.1

Натисніть Manage palette та встановіть вузол node-red-contrib-aedes

Налаштування брокера Aedes MQTT

Стандартний порт MQTT 1883, ці значення стандарті і їх змінювати не потрібно

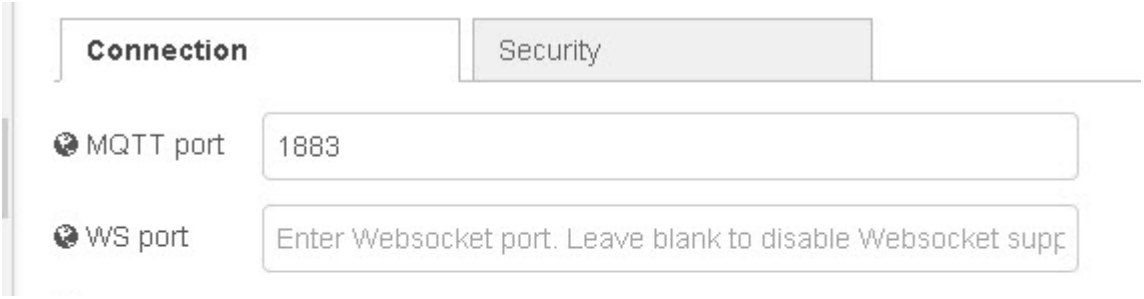


Рисунок 3.13 Налаштування вузла MQTT брокера

Inject -> function -> mqtt publish (симуляційні дані):

[39] У цій послідовності ми запрограмували періодичне введення фреймів JSON у такому форматі Inject вузол передає значення корисного навантаженння. У вузлі він пустий {} та приймає значення function node.

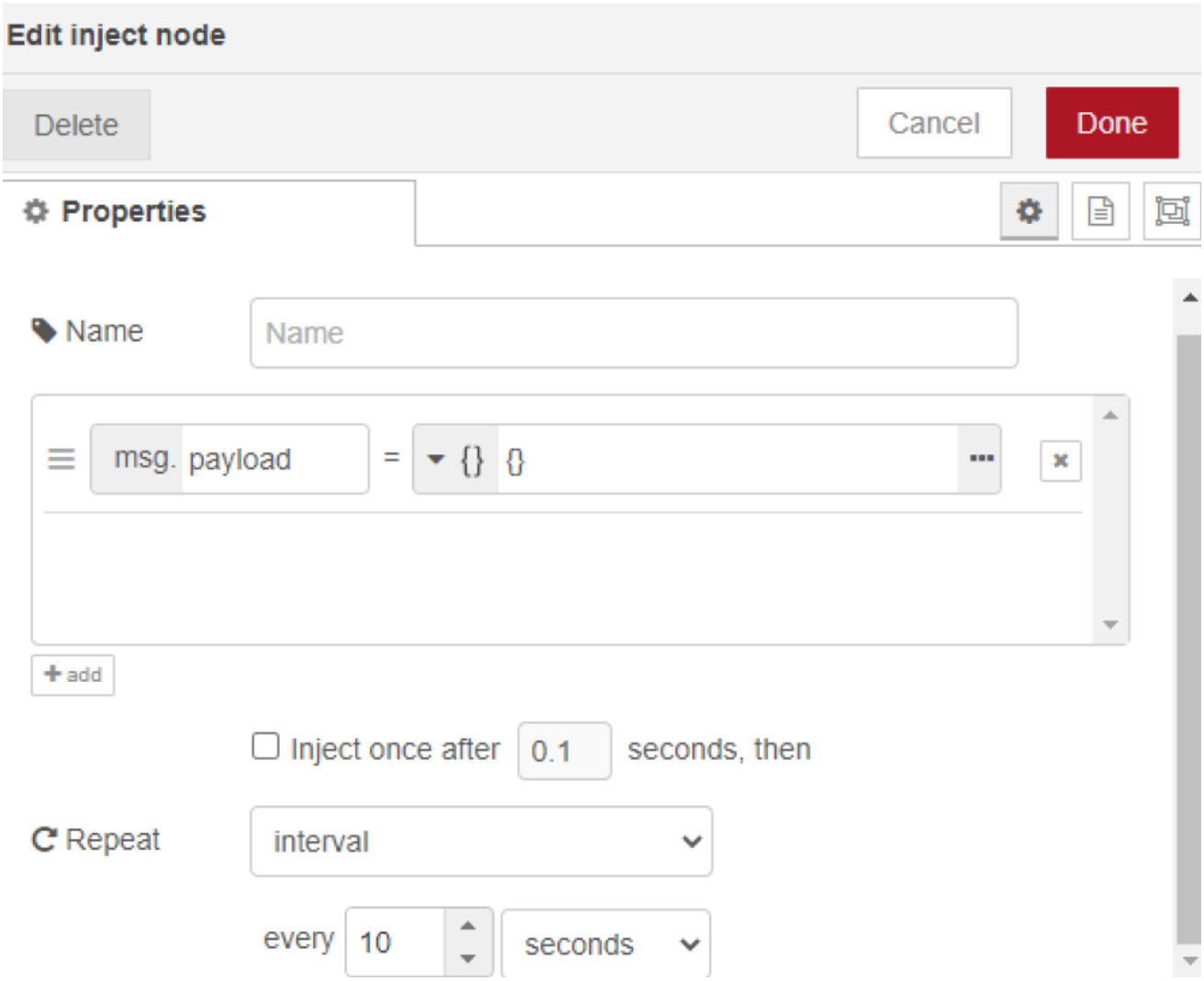


Рисунок 3.14 Inject node

Function node:

Даний вузол є функціональним і служить для потокової обробки даних.

[40] Цей вузол встановлює значення об'єкта msg.payload, яке містить три величини: температуру (temperature), вологість (humidity) та тиск (pressure). Значення генеруються випадковим чином за допомогою функції Math.random(). Зокрема:

- Температура генерується у діапазоні від -40 до 85 градусів Цельсія.
- Вологість генерується в діапазоні від 0 до 100%.
- Тиск генерується в діапазоні від 300 до 1110 гектопаскалей.

Отримані значення конвертуються в числа з плаваючою точкою (float) та округлюються до двох знаків після коми.

Після встановлення значень об'єкту msg.payload, вони повертаються через return msg, щоб їх можна було передати далі в потік обробки даних.

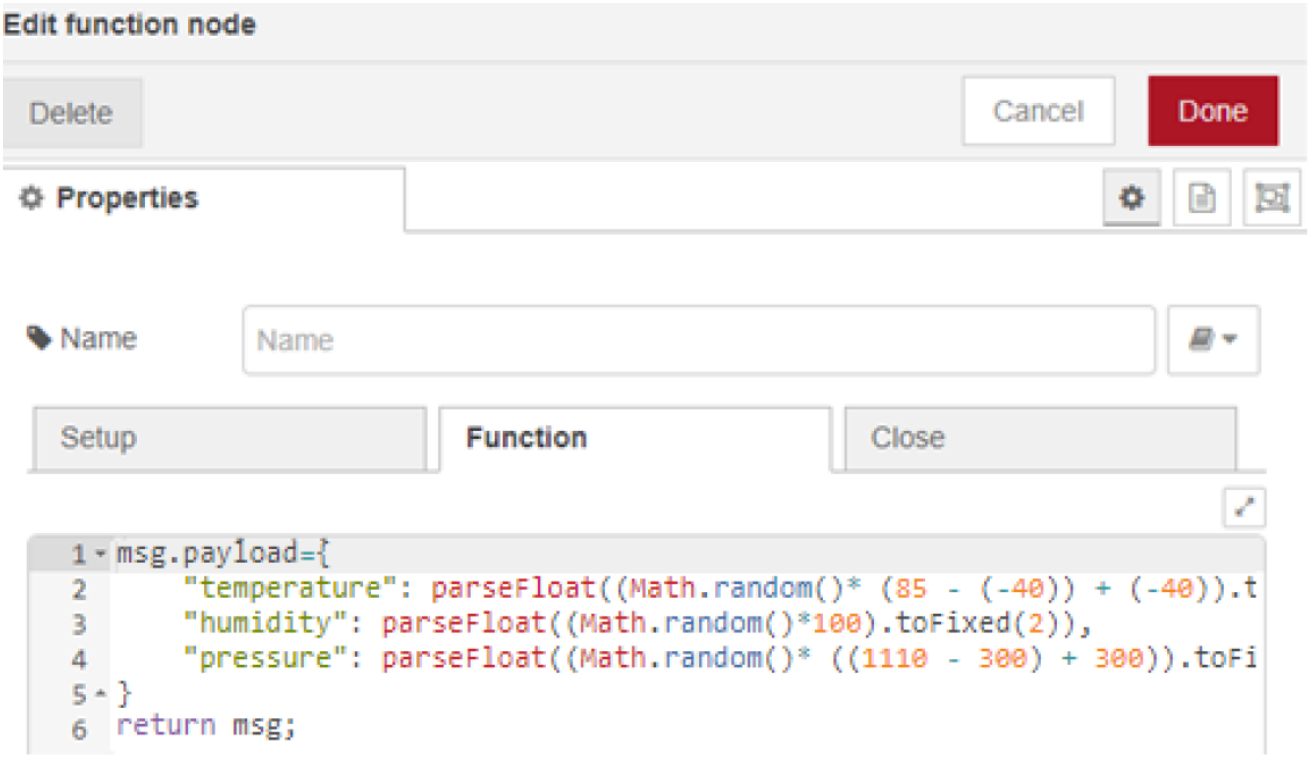


Рисунок 3.15 Налаштування симуляційних даних Function node

```
msg.payload={
"temperature": parseFloat((Math.random()* (85 - (-40)) + (-40)).toFixed(2)),
"humidity": parseFloat((Math.random()*100).toFixed(2)),
"pressure": parseFloat((Math.random()* ((1110 - 300) + 300)).toFixed(2))
}
return msg;
```

Після публікуємо ці JSON дані в темі MQTT

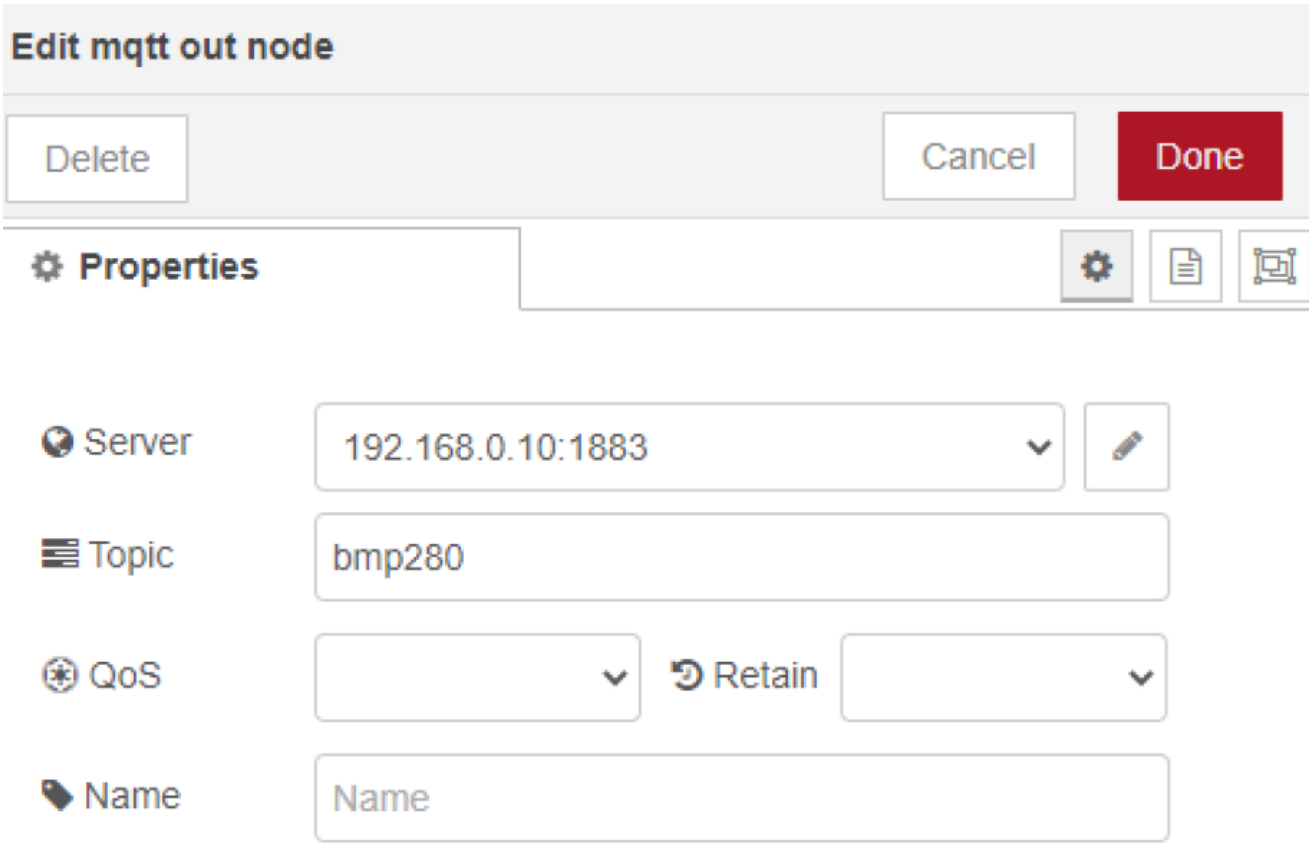


Рисунок 3.16 Налаштування вузла MQTT брокера

[41] Mqtt subscriber -> json -> Dashboards: (реальні чи симуляційні дані)

У цій послідовності ми підписалися на тему брокера MQTT і налаштували інформаційні панелі візуалізації.

Mqtt підписка:

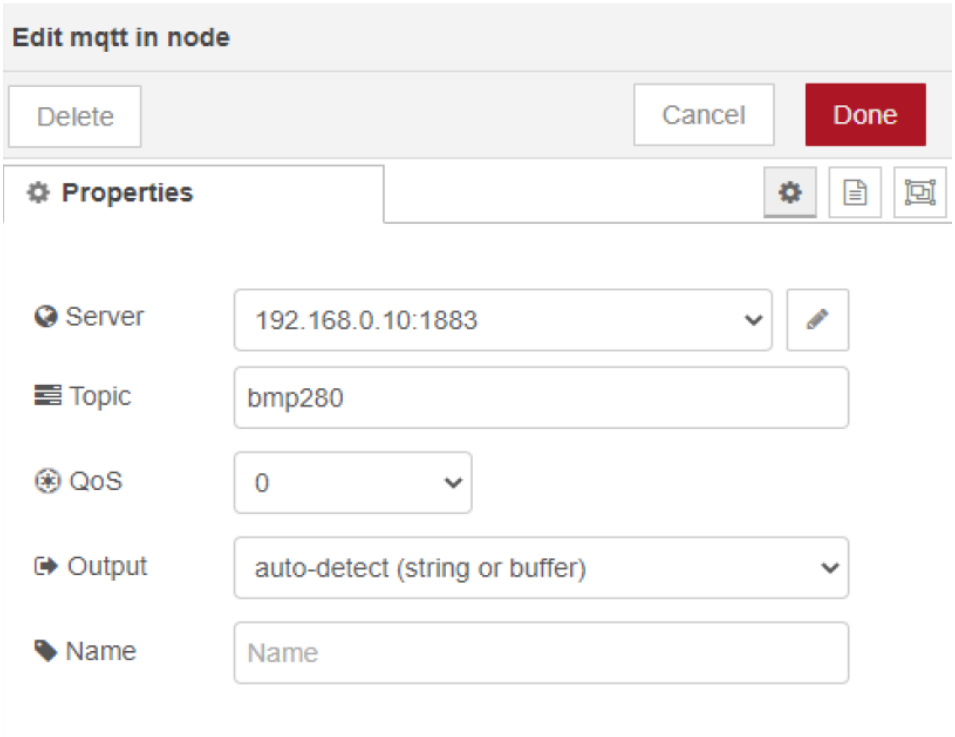


Рисунок 3.17 Mqtt підписка

[42] Вузол Json (Перетворення JSON string наObject):

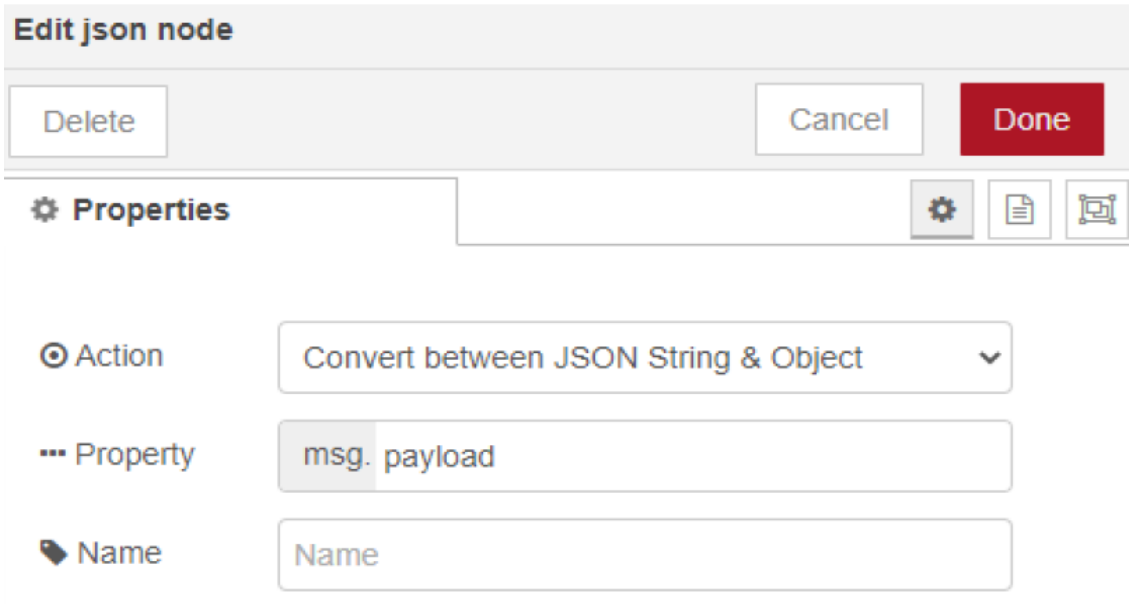


Рисунок 3.18 Вузол Json (Перетворення JSON string наObject)

Нарешті, ми налаштували інформаційну панель для трьох показників:

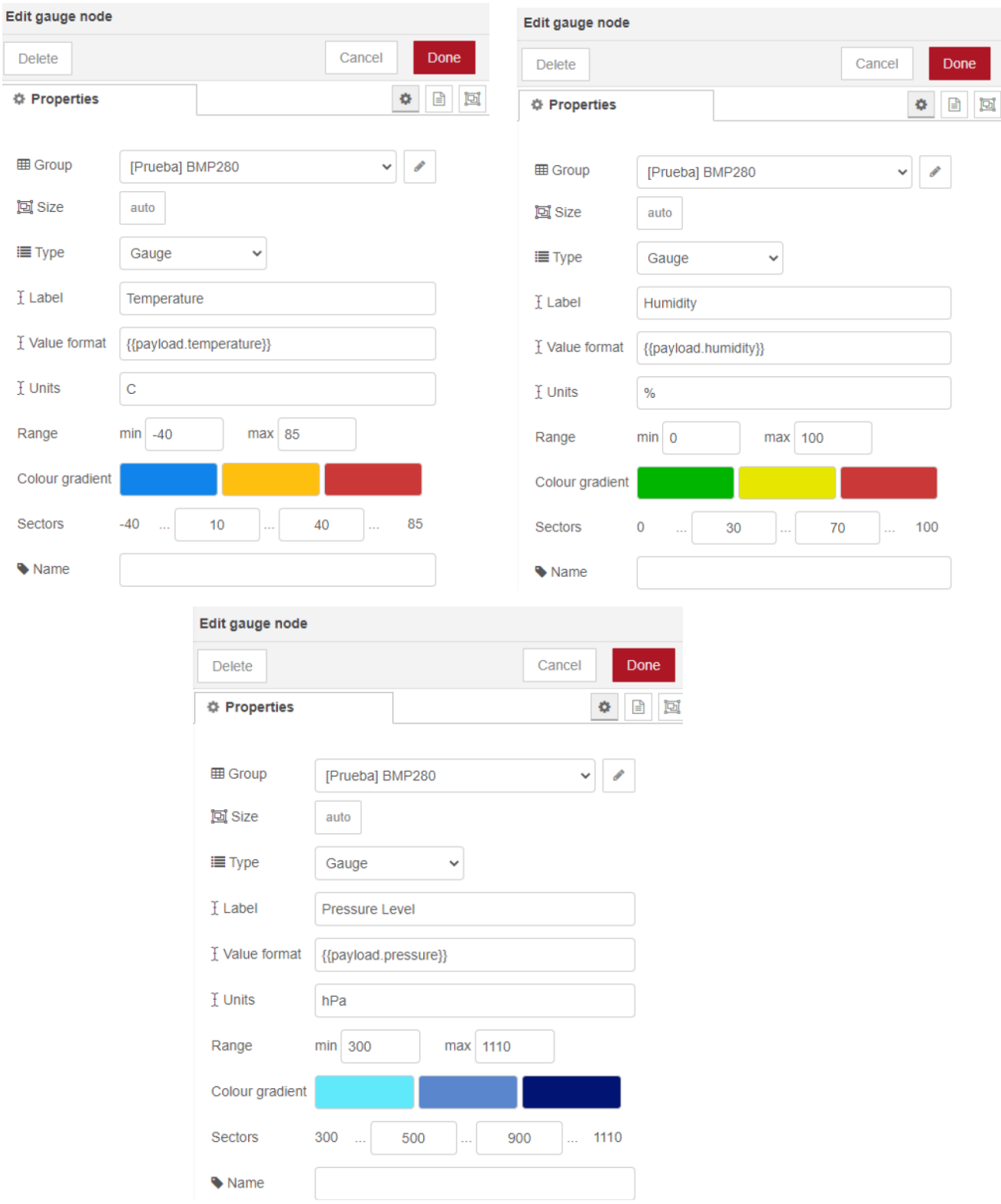


Рисунок 3.19 Node-RED налаштування панелі керування

Тест передачі даних

Прийом даних:

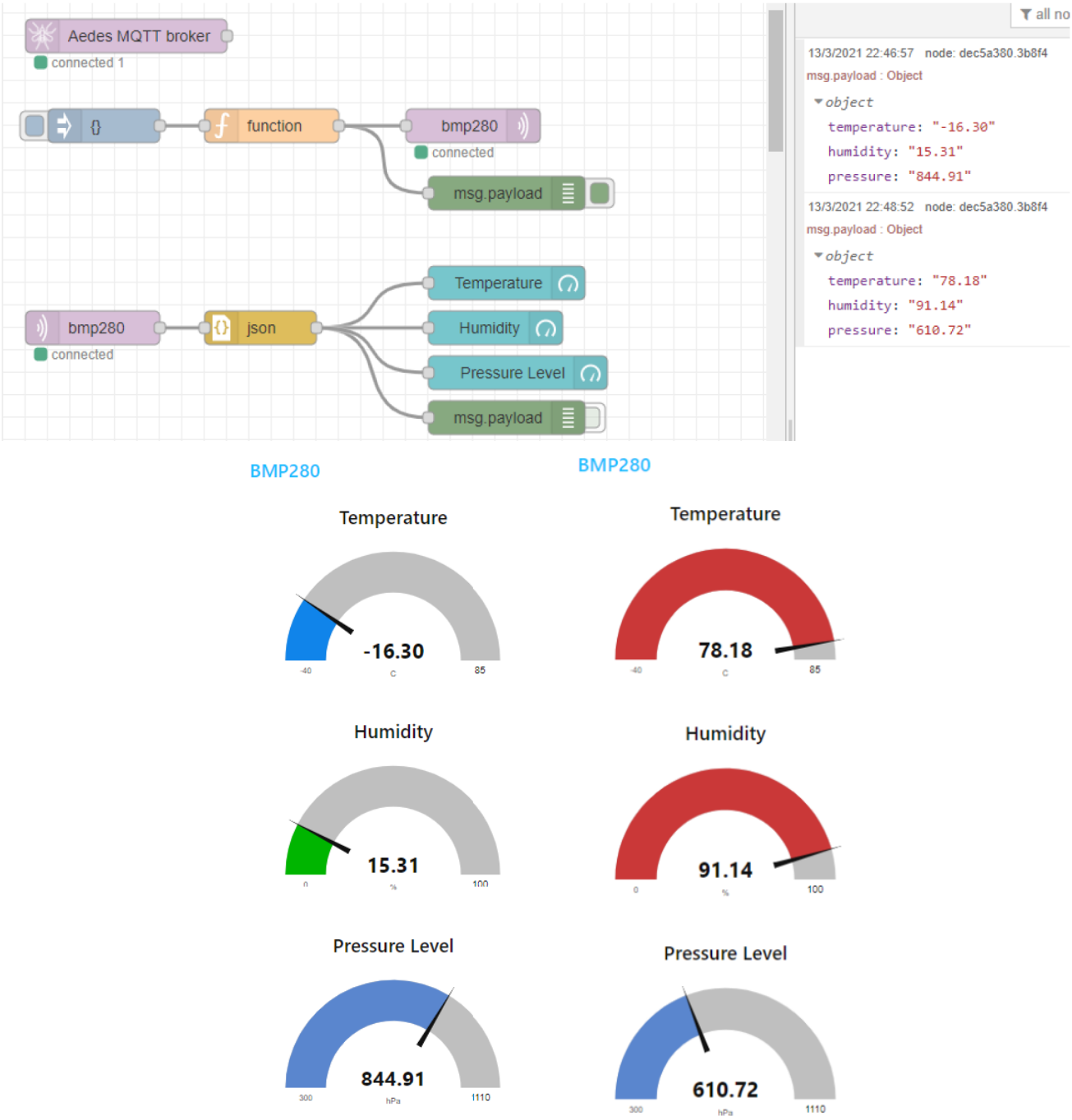


Рисунок 3.20 Тест передачі даних

3.3 Система моніторингу температури та вологості на основі сенсора BME280

[21] З’єднання ESP8266 з сенсором BME280

[43] Сенсорний модуль BME280 зчитує барометричний тиск, температуру та вологість. Оскільки тиск змінюється з висотою, ви також можете оцінити висоту. Існує кілька версій цього сенсорного модуля. У дослідженн використовуємо модуль, зображений на малюнку нижче.

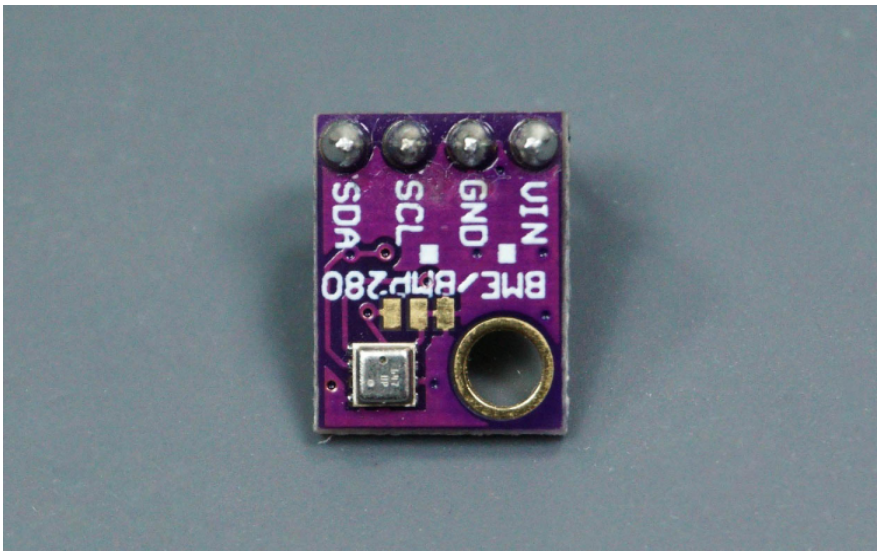


Рисунок 3.21 Сенсорний модуль BME280

[44] Цей датчик спілкується за допомогою протоколу зв’язку I2C, тому підключення дуже просте. Ви можете використовувати контакти ESP8266 I2C за замовчуванням, як показано в таблиці:

Таблиця 3.1 Схема зєднання виходів мікроконтролера ESP8266 та сенсора BME280

BME280	ESP8266
Vin	3.3V
GND	GND

SCL	GPIO 5
SDA	GPIO 4

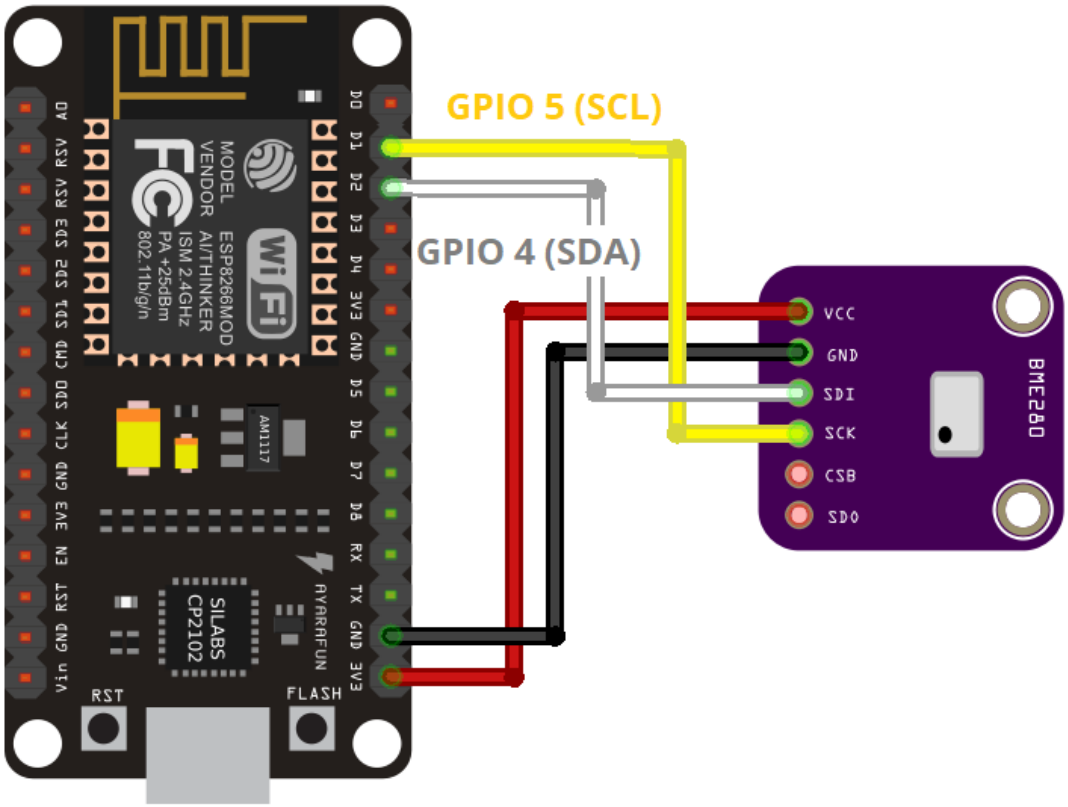


Рисунок 3.22 Схема зєднання виходів мікроконтролера ESP8266 та сенсора BME280

[45] Щоб отримати показники модуля датчика BME280, вам потрібно використовувати бібліотеку Adafruit_BME280. Виконайте наступні кроки, щоб установити бібліотеку в Arduino IDE:

Відкрийте Arduino IDE і перейдіть до **Sketch** > **Include Library** > **Manage Libraries** . Має відкритися Менеджер бібліотеки. Знайдіть « **adafruit bme280** » у полі пошуку та встановіть бібліотеку.

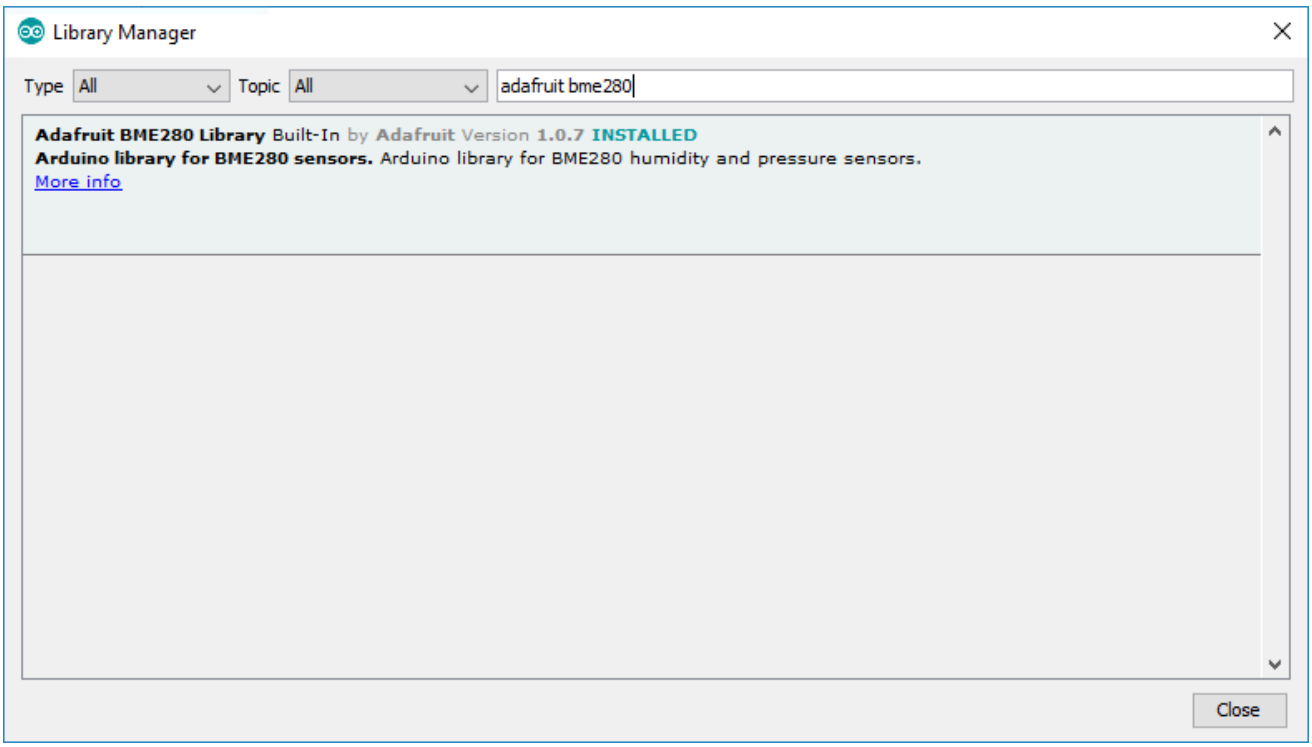


Рисунок 3.23 Менеджер бібліотеки

Код ESP8266 & BMP280 для Node-RED

```
        #include <Wire.h>
#include <SPI.h>
#include <WiFi.h> // include WiFi library
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#define BME_SCK 13
#define BME_MISO 12
#define BME_MOSI 11
#define BME_CS 10
// configure Sea level pressure
//https://www.meteo.cat/observacions/xema/dades
#define SEALEVELPRESSURE_HPA (1017.9)
#include <PubSubClient.h>
Adafruit_BME280 bme; // I2C
```

```
//YOU MUST CHANGE ssid
const char* ssid = "your_wifi_network"; // The SSID (name) of the Wi-Fi network you want to connect to
//YOU MUST CHANGE password
const char* password = "your_wifi_password"; // The password of the Wi-Fi network
//Platform server
const char* mqttServer = "192.168.43.79";
//Platform port
const int mqttPort = 1883;
//YOU MUST CHANGE your device TOKEN
const char* mqttUser = "";
const char* mqttPassword = "";
WiFiClient espClient;
PubSubClient client(espClient);
int status = WL_IDLE_STATUS;
unsigned long lastSend;
void setup() {
Serial.begin(115200); // Start the Serial communication to send messages to the computer
93
delay(10);
Serial.println('\n');
bool status;
// default settings
// (you can also pass in a Wire library object like &Wire2)
status = bme.begin(0x76);
if (!status) {
Serial.println("Could not find a valid BME280 sensor, check wiring!");
```

```
while (1);
}
/*Connecting ESP32 to WiFi */
WiFi.begin(ssid, password); // Connect to the network
Serial.print("Connecting to ");
Serial.print(ssid);
while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
delay(500);
Serial.print('.');
}
Serial.println('\n');
Serial.println("Connection established!");
Serial.print("IP address:\t");
Serial.println(WiFi.localIP()); // Send the IP address of the ESP32 to the computer
Serial.println('\n');
//Connecting ESP32 to MQTT Broker, subscribing and receiving messages
client.setServer(mqttServer, mqttPort); //configure the PubSubClient with the
address and port
while (!client.connected()) {
Serial.println("Connecting to MQTT...");
if (client.connect("ESP32Client", mqttUser, mqttPassword )) {
Serial.println("Client connected");
} else {
Serial.print("failed with state ");
Serial.print(client.state());
delay(2000);
}
}
lastSend=0;
```

```
94
}
void loop() {
if(millis()-lastSend>1000){
getAndSendData();
lastSend=millis();
}
client.loop();
}
void getAndSendData()
{
Serial.print("getAndSendData: ");
float h=bme.readHumidity();
float t=bme.readTemperature();
float p=bme.readPressure()/ 100.0F;
float a=bme.readAltitude(SEALEVELPRESSURE_HPA);
String temperature=String(t);
String humidity=String(h);
String pressure=String(p);
String altitude=String(a);
Serial.print("Temperature = ");
Serial.print(temperature);
Serial.println(" *C");
Serial.print("Pressure = ");
Serial.print(pressure);
Serial.println(" hPa");
Serial.print("Approx. Altitude = ");
Serial.print(altitude);
Serial.println(" m");
```

```
Serial.print("Humidity = ");
Serial.print(humidity);
Serial.println(" %");
Serial.println();
//prepare a JSON payload string
String payload = "{";
payload += "\"temperature\":"; payload += temperature; payload +=",";
//payload += "\"temperature\":"; payload += temperature;
95
payload += "\"pressure\":"; payload += pressure; payload +=",";
payload += "\"humidity\":"; payload += humidity; payload +=",";
payload += "\"altitude\":"; payload += altitude;
payload += "}";
char telemetry[1000];
payload.toCharArray( telemetry, 1000 );
client.publish( "bmp280", telemetry );
Serial.println( telemetry );
Serial.println('\n');
delay(10000);
}
```

ESP8266 & LED для Node-RED

```
#include <WiFi.h> // include WiFi library
#include <PubSubClient.h> // Include library to create a client MQTT
const char* ssid = "your_wifi_network"; // The SSID (name) of the Wi-Fi network
you want to connect to
const char* password = "your_wifi_password"; // The password of the Wi-Fi
network
```

```
const char* mqttServer = "192.168.43.79"; // The IP address of your MQTT Broker
const int mqttPort = 1883; // The TCP port of your MQTT Broker
const char* mqttUser = ""; // MQTT client user
const char* mqttPassword = ""; // MQTT client password
const int ledPin = 4;
WiFiClient espClient_1; // Define the client that allows us to transmit by WiFi. Must be different for each MQQT client that you want to connect to a MQTT Broker
PubSubClient client(espClient_1); // Define the MQTT client that will access the MQTT Broker.
char payload[512]; // It will be the message to RECEIVE
/* Look for message received */
void callback(char* topic, byte* payload, unsigned int length) {
Serial.print("Message arrived in topic: ");
Serial.println(topic);
Serial.print("Message:");
96
for (int i = 0; i < length; i++) {
Serial.print((char)payload[i]);
}
digitalWrite (ledPin, HIGH);
delay(500);
digitalWrite (ledPin, LOW);
delay(500);
}
void setup() {
Serial.begin(115200); // Start the Serial communication to send messages to the computer
delay(10);
Serial.println('\n');
```

```
pinMode (ledPin, OUTPUT);
/*Connecting ESP32 to WiFi */
WiFi.begin(ssid, password); // Connect to the network
Serial.print("Connecting to ");
Serial.print(ssid);
while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
delay(500);
Serial.print('.');
}
Serial.println('\n');
Serial.println("Connection established!");
Serial.print("IP address:\t");
Serial.println(WiFi.localIP()); // Send the IP address of the ESP32 to the computer
/*Connecting ESP32 to MQTT Broker, subscribing and receivin messages */
client.setServer(mqttServer, mqttPort); //configure the PubSubClient with the
address and port
client.setCallback(callback); // for receiving messages
while (!client.connected()) { // While mqtt client it's not connected stay at loop
Serial.println("Trying to connect to MQTT");
97
if (client.connect("espClient_1", mqttUser, mqttPassword )) { // connect to broker
(return true if connect, false if not)
Serial.println("connected");
if(client.subscribe("alert", 1)){
Serial.println("Subscribed to alert");
}
} else {
Serial.print("failed with state ");
Serial.print(client.state());
```

```
delay(2000);
}
}
}
void loop() {
client.loop();
}
```

3.4 Розгортання InfluxDB та Grafana

Встановлення InfluxDB

[22] Встановимо базу даних часових рядів InfluxDB:

docker pull influxdb:latest

mkdir /docker/influxdb

після завантаження образу можна запустити контейнер InfluxDB:

docker run -itd --name influxdb --restart=unless-stopped -p 8086:8086 -v /docker/influxdb:/var/lib/influxdb influxdb:1.8.5

Наступним кроком є створення бази даних і користувача, який матиме доступ до цієї бази даних. Спочатку запустіть InfluxDB CLI:

docker exec -it influxdb influx

Далі створимо базу даних і користувача:

> create database home

```
postgres=# CREATE DATABASE thingsboard;
CREATE DATABASE
postgres=# show databases
postgres-# \l
                                  List of databases
    Name     |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-------------+----------+----------+-------------+-------------+-----------------------
 postgres    | postgres | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 |
 template0   | postgres | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 | =c/postgres          +
             |          |          |             |             | postgres=CTc/postgres
 template1   | postgres | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 | =c/postgres          +
             |          |          |             |             | postgres=CTc/postgres
 thingsboard | postgres | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 |
(4 rows)
```

Рисунок 3.24 створення бази даних InfluxDB

> use home

Домашня сторінка використання бази даних

> create user pi with password 'passwordhere' with all privileges

> grant all privileges on home to pi

> show users

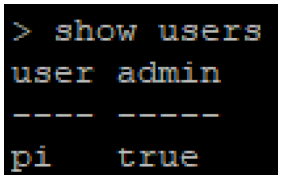


Рисунок 3.25 перегляд користувачів бази даних InfluxDB

> exit

За допомогою цих кількох рядків ми створили базу даних під назвою thingboard і користувача з іменем користувача pi та паролем passwordhere.

Встановлення Grafana

Встановимо веб-додаток для аналітики та інтерактивної візуалізації Grafana, як альтернативу інформаційній панелі Node-RED, де взаємозв’язок із базою даних набагато легший.

docker pull grafana/grafana:latest

cd /

sudo mkdir /docker/grafana-storage

docker run -itd --name grafana --user 0 --restart=unless-stopped - v /docker/grafana-storage/data:/var/lib/grafana -p 3000:3000 grafana/grafana

Графана працює на машині та доступна з будь-якого пристрою в локальній мережі.

Перший вхід Grafana

[23] Після запуску контейнеру Grafana, система стає доступною через веб-браузер. Перейдіть за адресою http://localhost:3000, якщо Grafana встановлена локально, або до відповідної адреси сервера.

Вхід в систему:

Введіть ім'я користувача та пароль за замовчуванням (зазвичай admin/admin) для входу до системи Grafana вперше.

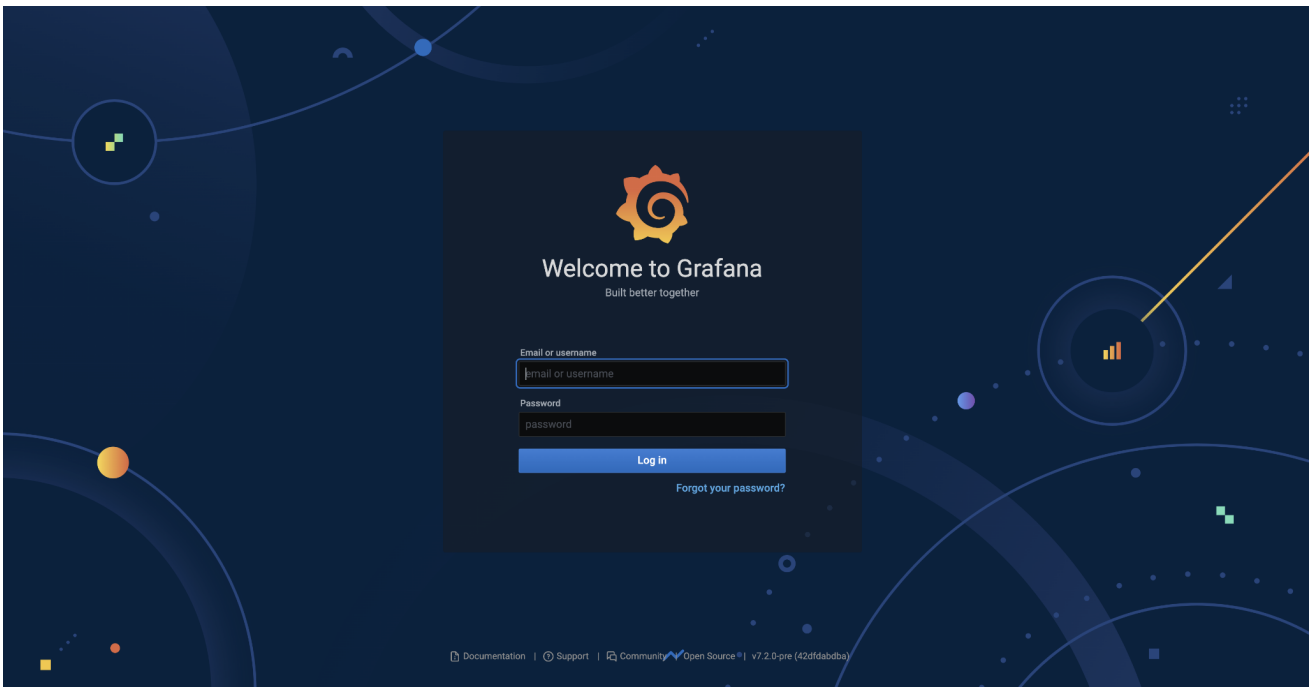


Рисунок 3.26 Стартовий екран Grafana

Додавання джерела даних:

Для підключення джерела даних до InfluxDB, перейдіть до вкладки "Configuration" (Конфігурація) та оберіть "Data Sources" (Джерела даних).

Натисніть "Add data source" (Додати джерело даних) і оберіть "InfluxDB" як тип джерела.

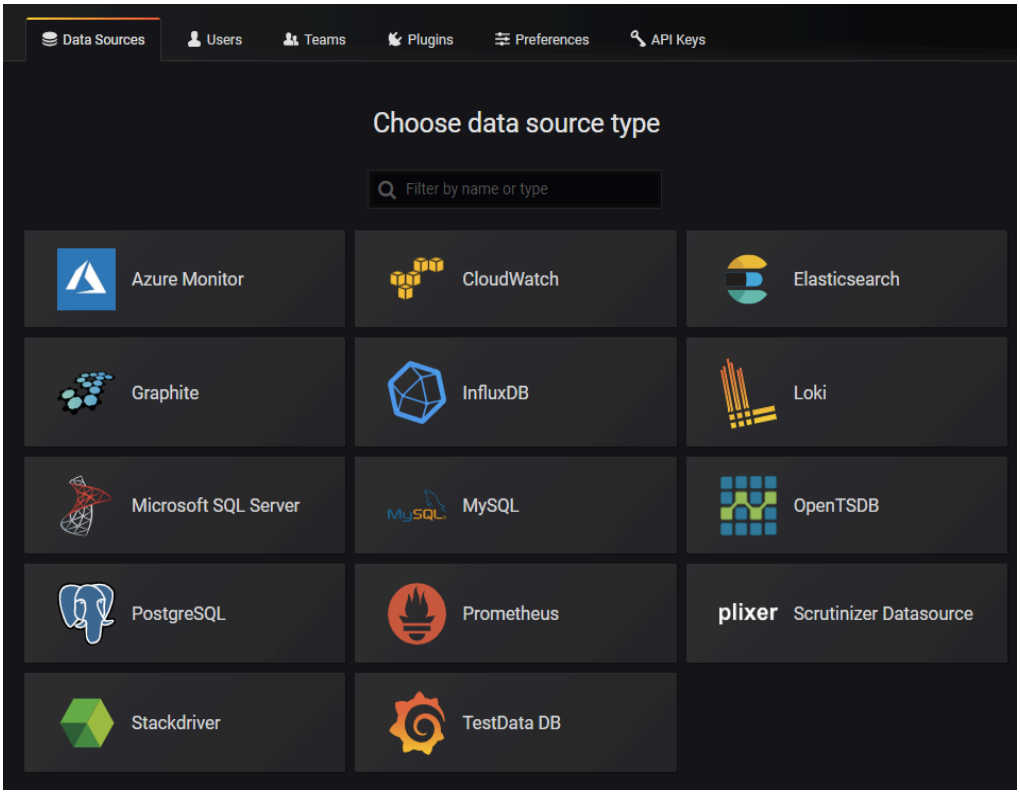


Рисунок 3.27 Джерела даних Grafana

Налаштуйте параметри підключення до InfluxDB, включаючи URL, ім'я бази даних, ім'я користувача та пароль.

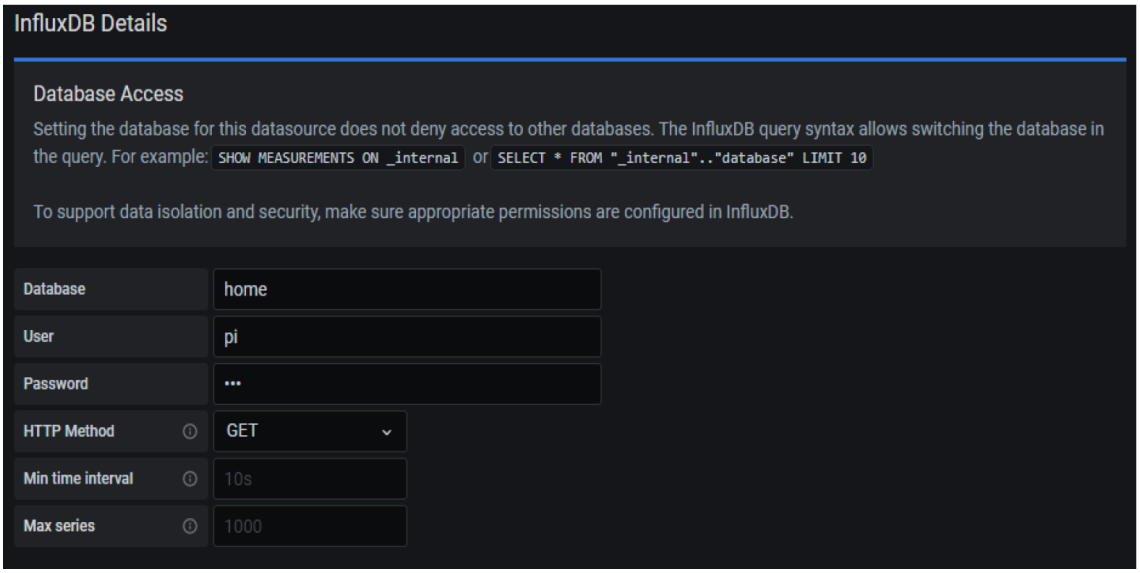


Рисунок 3.28 Налаштування параметрів підключення до InfluxDB

Налаштування InfluxDB у Node RED

[24] Встановіть модуль influxdb:

У Node-Red необхідно натиснути Manage Pelete -> знайти та встановити вузол «node-red-contrib-influxdb»

Після встановлення вузол буде доступнийм на полі Node-Red ліворуч.

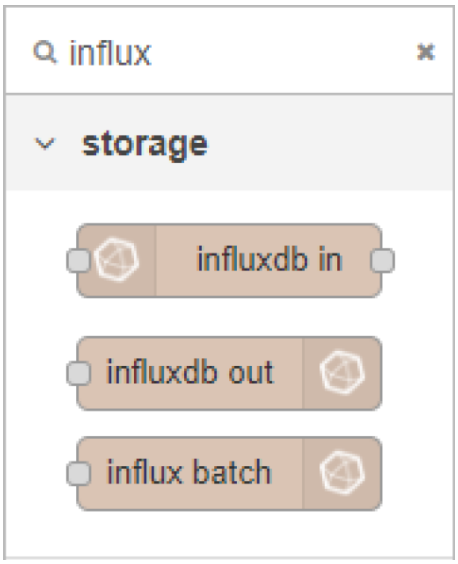


Рисунок 3.29 Налаштування вузлів прийому даних InfluxDB

Щоб імпортувати імпортувати дані з ESP8266, що передаються в тему MQTT «bmp280», і передати дані в базу даних Influx відредагуємо параметри вузла [49]

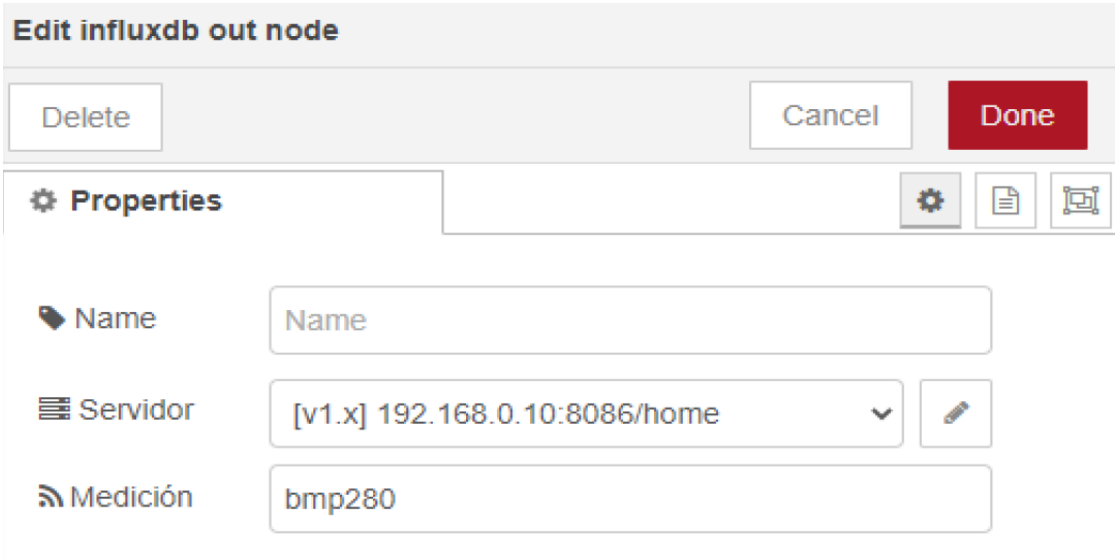


Рисунок 3.30 налаштування вузла Node-Red

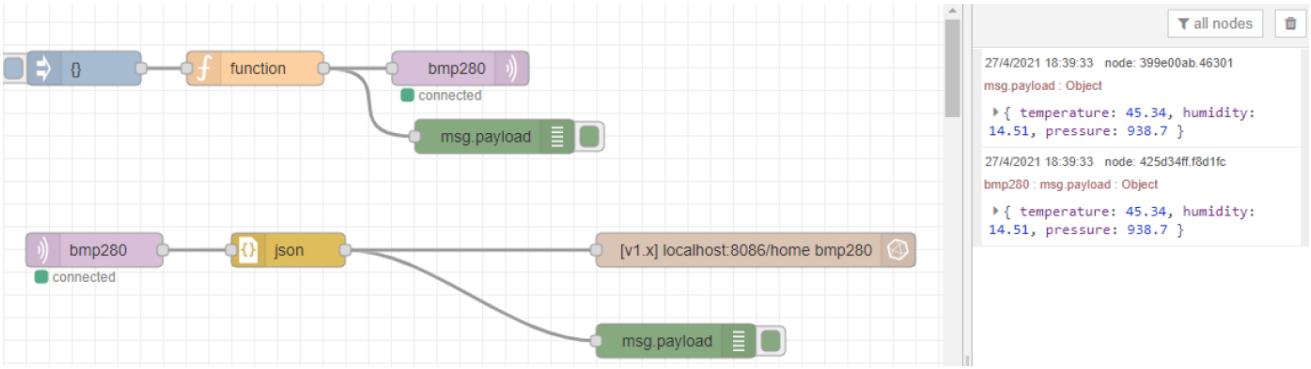


Рисунок 3.31 Прийом реальних даних у Node-RED

Ми також можемо переглядати ці данні в базі даних InfluxDB

```
> SELECT * FROM bmp280 LIMIT 10
name: bmp280
time                humidity pressure temperature
----                -------- -------- -----------
1619541574000000000 14.51    938.7    45.34
```

Рисунок 3.32 Дані з бази даних influxDB

Примітка: щоб видалити деякі дані з цієї бази даних, використаємо наступну команду в оболонці:

Видалити всю базу даних:

DELETE FROM table_name WHERE condition

Видалити стовпець:

ALTER TABLE table_name DROP COLUMN column_name

Видалити певні дані:

DELETE FROM table_name WHERE column_name=specific_data

Node-RED Flow

Крім представлення у вигляді вузлів, конфігурацію вузлів легко представити у вигляді текстовому JSON вигляді. Це дозволяє легше та простіше копіювати та переносити конфігурацію між різними хостами. [50]

[{"id":"f8e5facc.9be588","type":"tab","label":"Final Flow","disabled":false,"info":""},{"id":"2ed4ab5c.22b614","type":"mqtt in","z":"f8e5facc.9be588","name":"","topic":"bmp280","qos":"2","datatype":"auto","broker":"83fc0f7f.59a84","x":119,"y":220,"wires":[["b3932d8c.01ed7"]]},{"id":"c74cdcd.8eaa02","type":"influxdb out","z":"f8e5facc.9be588","influxdb":"48d8c90b.301e38","name":"","measurement":"bmp280","precision":"s","retentionPolicy":"","database":"database","precisionV18FluxV20":"ms","retentionPolicyV18Flux":"","org":"organisation","bucket":"bucket","x":699,"y":220,"wires":[]},{"id":"425d34ff.f8d1fc","type":"debug","z":"f8e5facc.9be588","name":"","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"false","statusVal":"","statusType":"auto","x":629,"y":300,"wires":[]},{"id":"b3932d8c.01ed7","type":"json","z":"f8e5facc.9be588","name":"","property":"payload","action":"","pretty":false,"x":289,"y":220,"wires":[["425d34ff.f8d1fc","c74cdcd.8eaa02"]]},{"id":"647d8be1.d71d24","type":"function","z":"f8e5facc.9be588","name":"","func":"msg.payload={\n \"temperature\": parseFloat((Math.random()* (85 - (-40)) + (-40)).toFixed(2)),\n \"humidity\": parseFloat((Math.random()*100).toFixed(2)),\n \"pressure\": parseFloat((Math.random()* ((1110 - 300) + 300)).toFixed(2))\n}\nreturn msg;\n","outputs":1,"noerr":0,"initialize":"","finalize":"","x":260,"y":60,"wires":[["399e00ab.46301","3129be12.876422"]]},{"id":"5d886fec.29f6c","type":"inject","z":"f8e5facc.9be588","name":"","props":[{"p":"payload"}],"repeat":"","crontab":"","once":false,"onceDelay":0.1,"topic":"","payload":"{}","payloadType":"json","x":110,"y":60,"wires":[["647d8be1.d71d24"]]},{"id":"3129be12.876422","type":"mqtt

out","z":"f8e5facc.9be588","name":"","topic":"bmp280","qos":"","retain":"","broker":"83fc0f7f.59a84","x":440,"y":60,"wires":[]},{"id":"399e00ab.46301","type":"debug","z":"f8e5facc.9be588","name":"","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","targetType":"msg","statusVal":"","statusType":"auto","x":470,"y":120,"wires":[]},{"id":"83fc0f7f.59a84","type":"mqtt-broker","name":"","broker":"localhost","port":"1883","clientid":"","usetls":false,"compatmode":false,"keepalive":"60","cleansession":true,"birthTopic":"","birthQos":"0","birthPayload":"","closeTopic":"","closeQos":"0","closePayload":"","willTopic":"","willQos":"0","willPayload":""},{"id":"48d8c90b.301e38","type":"influxdb","hostname":"localhost","port":"8086","protocol":"http","database":"home","name":"","usetls":false,"tls":"","influxdbVersion":"1.x","url":"http://localhost:8086","rejectUnauthorized":true}]

Збір даних за допомогою Grafana

Створюючи новий дашборд і панель, ми можемо спостерігати за показниками сенсора. Проте спочатку потрібно налаштувати запити, які будуть надсилатись до бази данних Influxdb.

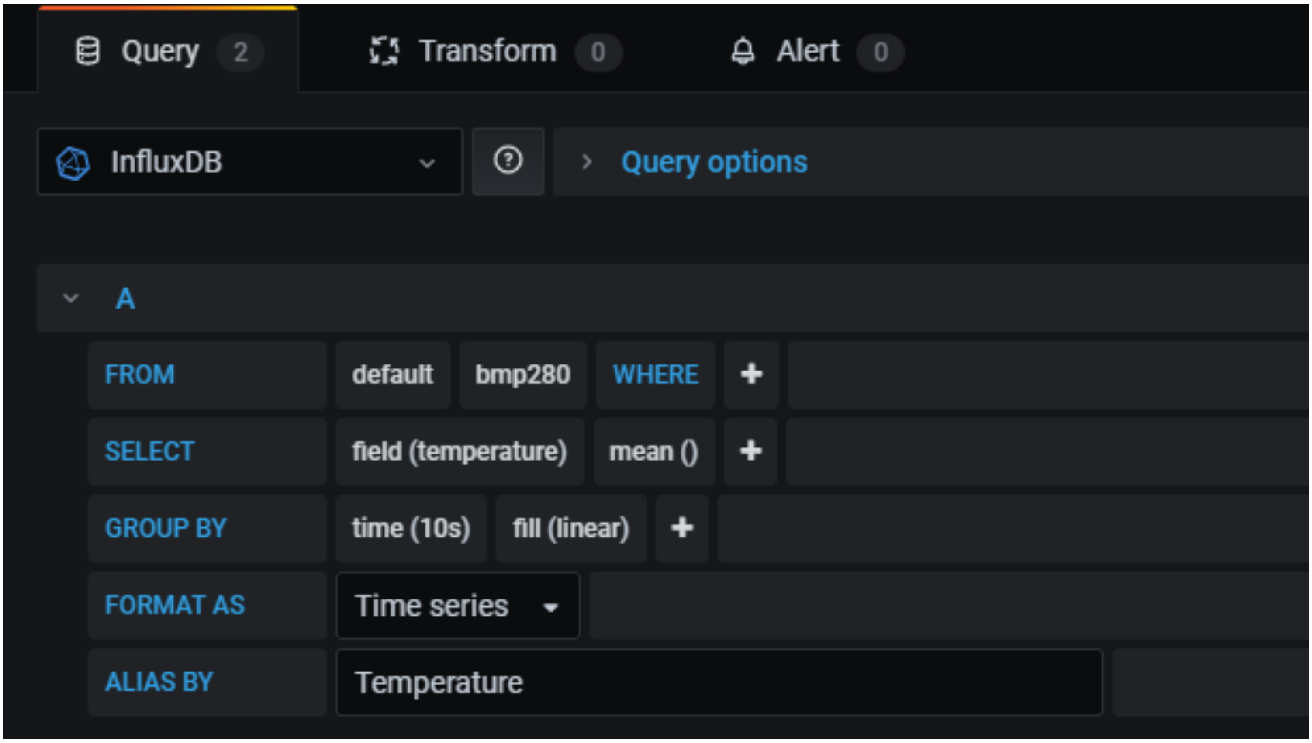


Рисунок 3.33 Створення дашборду Grafana

[25] Крім того, праворуч ми можемо детальніше налаштувати панель (змінювати осі, дисплей, легенду, назву, кольори, заливку, градієнт тощо).

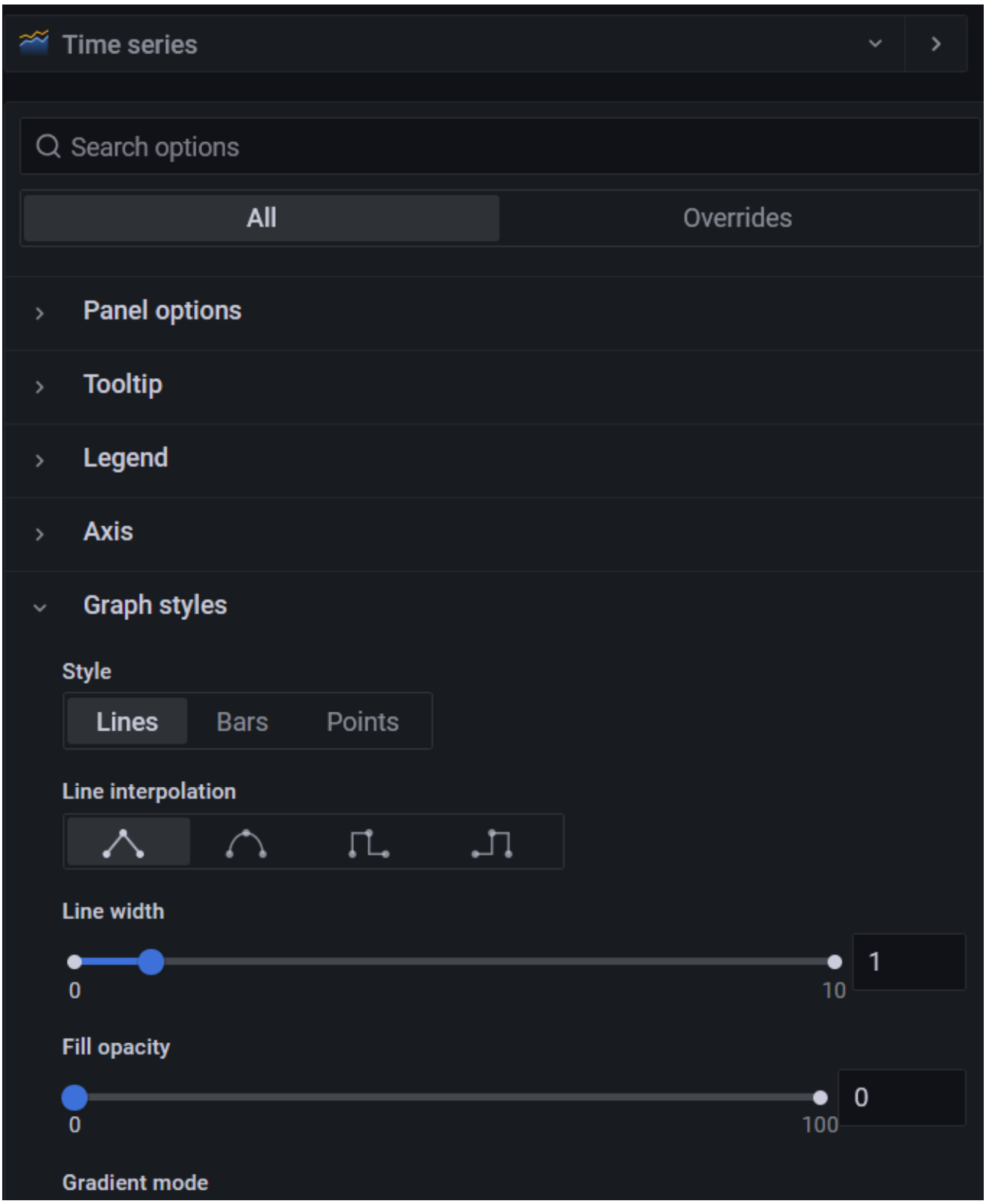


Рисунок 3.34 детальне налаштування панелі

В результаті маємо графік для візуалізації даних про температуру та вологість нашого сенсора:

Рисунок 3.35 Створення дашборду Grafana

Рисунок 3.36 Графіки Grafana

Створення панелей та графіків:

[26] Перейдіть до вкладки "Create" (Створити) та оберіть "Dashboard" (Дошка), яку ви хочете створити.

Виберіть створений вами Dashboard та натисніть "Add new panel" (Додати нову панель).

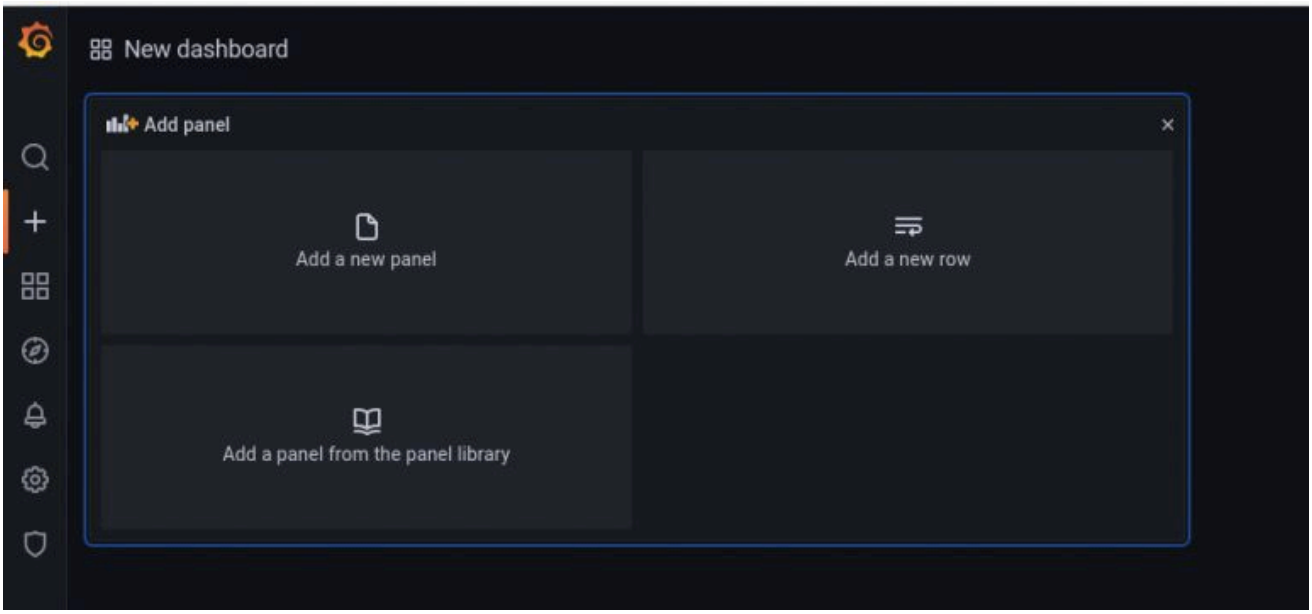


Рисунок 3.37 Вкладка додати нову панель

Виберіть тип графіка або візуалізації, яку ви хочете використовувати для відображення MQTT даних.

У налаштуваннях панелі виберіть джерело даних (InfluxDB) та налаштуйте запити для отримання даних з бази даних InfluxDB. Збережіть ваш Dashboard та опублікуйте його, щоб інші користувачі мали до нього доступ.

Конфігурація сповіщень:

[27] Grafana використовує менеджер сповіщень Alertmanagers для надсилання сповіщень про активацію та вирішені сповіщення. Grafana має власний Alertmanager, якийпідтримує надсилання сповіщень з інших Alertmanager, таких як Prometheus. Для сповіщень Grafana використовує політики і контактні точки, щоб налаштувати як часто потрібно надсилати повідомлення, чи слід усі сповіщення надсилати в одному сповіщенні, надсилати в згрупованих сповіщеннях, на основі набору міток, чи як окремі сповіщення.

Тепер ваш інтерфейс Grafana готовий до візуалізації та моніторингу MQTT даних. Ви можете відстежувати стан системи та аналізувати дані на графіках та панелях.

Grafana надає потужні можливості для візуалізації та моніторингу MQTT даних, і ви можете створювати різноманітні графіки та панелі, які відображають ваші дані в зручному для аналізу способі.

Використання системних ресурсів Raspberry Pi

Цей проект був встановлений і протестований на 16 ГБ пам'яті Micro SD.

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        14G  2.6G   11G  20% /
devtmpfs        430M     0  430M   0% /dev
tmpfs           463M     0  463M   0% /dev/shm
tmpfs           463M  6.5M  456M   2% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           463M     0  463M   0% /sys/fs/cgroup
/dev/mmcblk0p1  253M   48M  205M  19% /boot
tmpfs            93M     0   93M   0% /run/user/1000
overlay          14G  2.6G   11G  20% /var/lib/docker/overlay2/7f6da8ablefc729b
overlay          14G  2.6G   11G  20% /var/lib/docker/overlay2/4f6aa62583c01ca8c
overlay          14G  2.6G   11G  20% /var/lib/docker/overlay2/612c9f15d23362e81
overlay          14G  2.6G   11G  20% /var/lib/docker/overlay2/9e099dca0c875c7fe
```

Рис. 3.38 Використання пам’яті Raspberry Pi

На малюнку видно, обсяг пам’яті, який використовується для збору даних кожні 10 секунд. Споживання пам’яті дуже мале, враховуючи встановлене програмне забезпечення, тому для заповнення пам’яті знадобиться принаймні кілька років. (Це обчислюється без видалення та усереднення старих збережених даних, що можна зробити за допомогою InfluxDB).

Тест з'єднань MQTT

Щоб перевірити здатність прийому даних встановленого докера MQTT, ми виконали порівняльний тест із різною кількістю клієнтів і надісланих повідомлень, щоб побачити їх швидкість[46][47][48].

Ось результати сценарію з кількома контрольними тестами (mqtt-stresser & mqtt-bench), у якому 10 клієнтів надсилають кілька повідомлень (100 повідомлень/на клієнта), і перевіряються 5 разів:

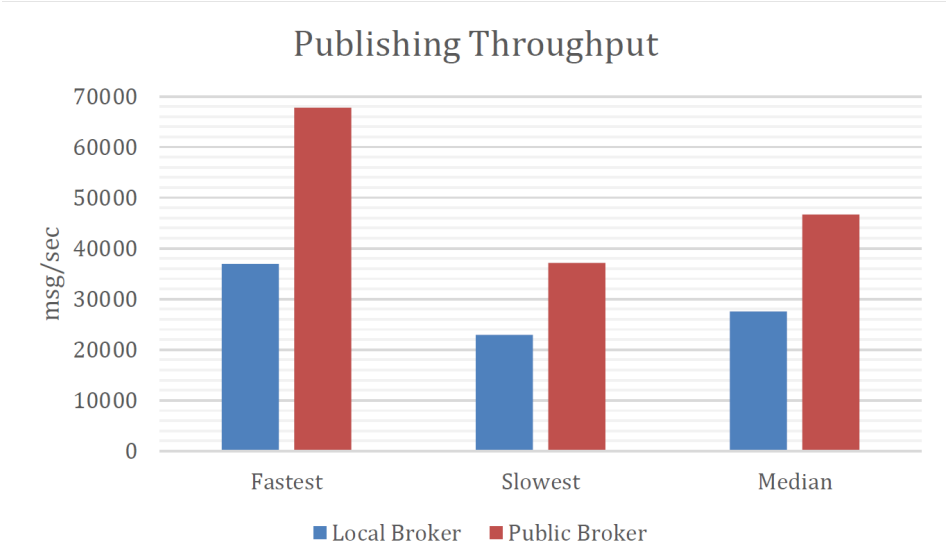


Рис. 3.39 Mqtt-stresser результати публікації повідомлень клієнтів 10, Повідомлень 100

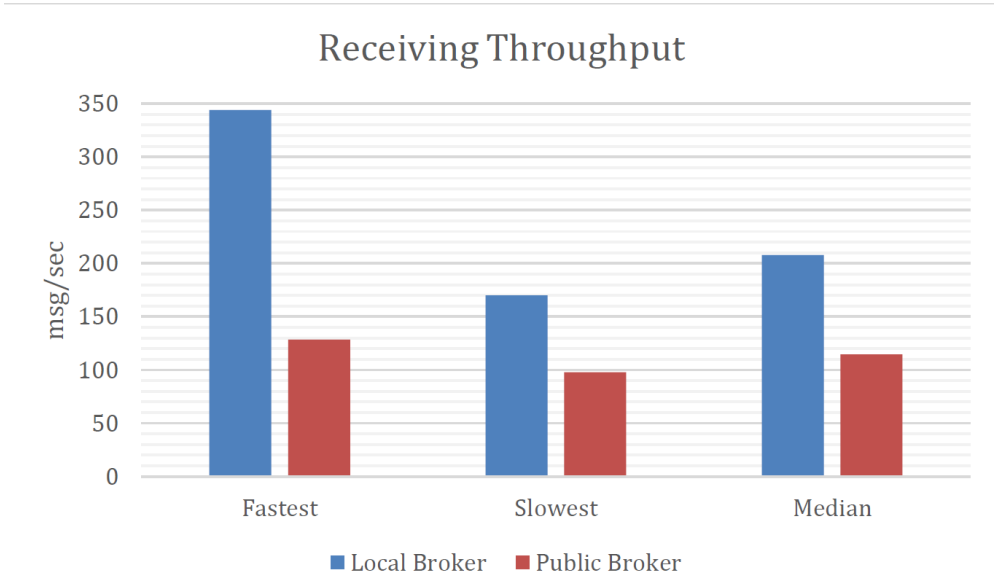


Рис. 3.40 Результати отримання Mqtt-stresser: клієнтів 10, Повідомлень 100

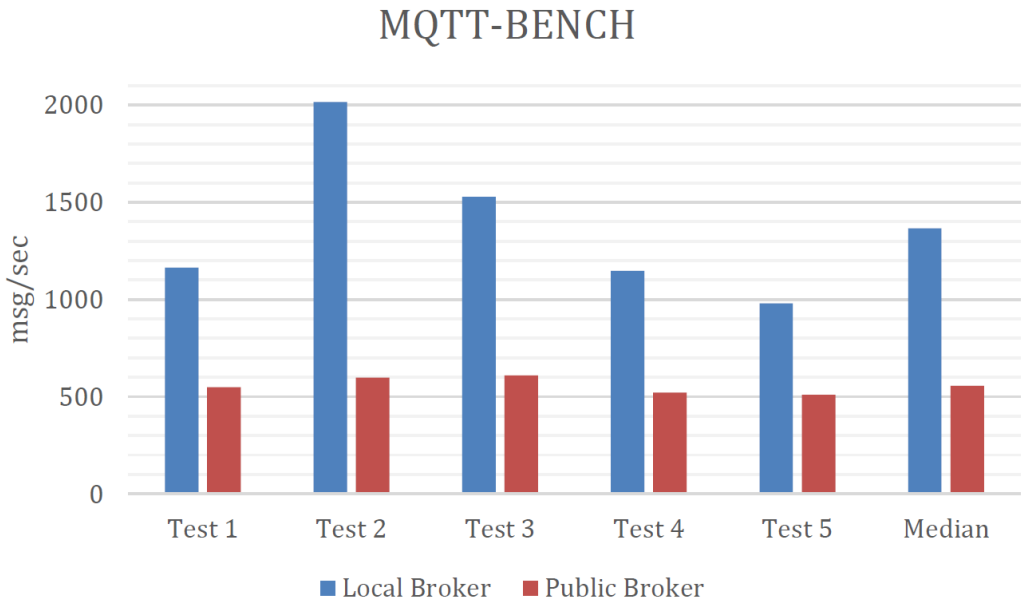


Рис. 3.41 Результати отримання Mqtt-bench: клієнтів 10, Повідомлень 100

Симулюючи за допомогою тесту mqtt-bench, з 10 одночасними клієнтами MQTT, кожен з яких надсилає 100 повідомлень по 1 Кбайт кожне, ми бачимо, як ми можемо отримати приблизно 1000 повідомлень на секунду, отримані результати:

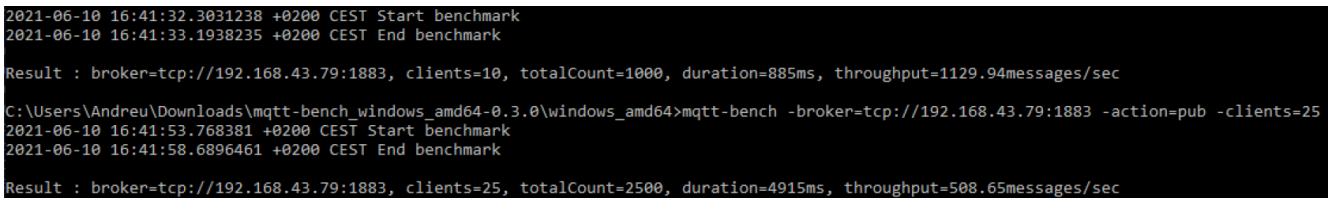


Рис. 3.42 Приклад команди mqtt-bench

З іншого боку, якщо ми збільшимо кількість постійних клієнтів до 25, ми побачимо, як кількість повідомлень за секунду радикально зменшується менше ніж наполовину.

Вплив такої великої кількості повідомлень не викликає значного збільшення споживання ЦП:

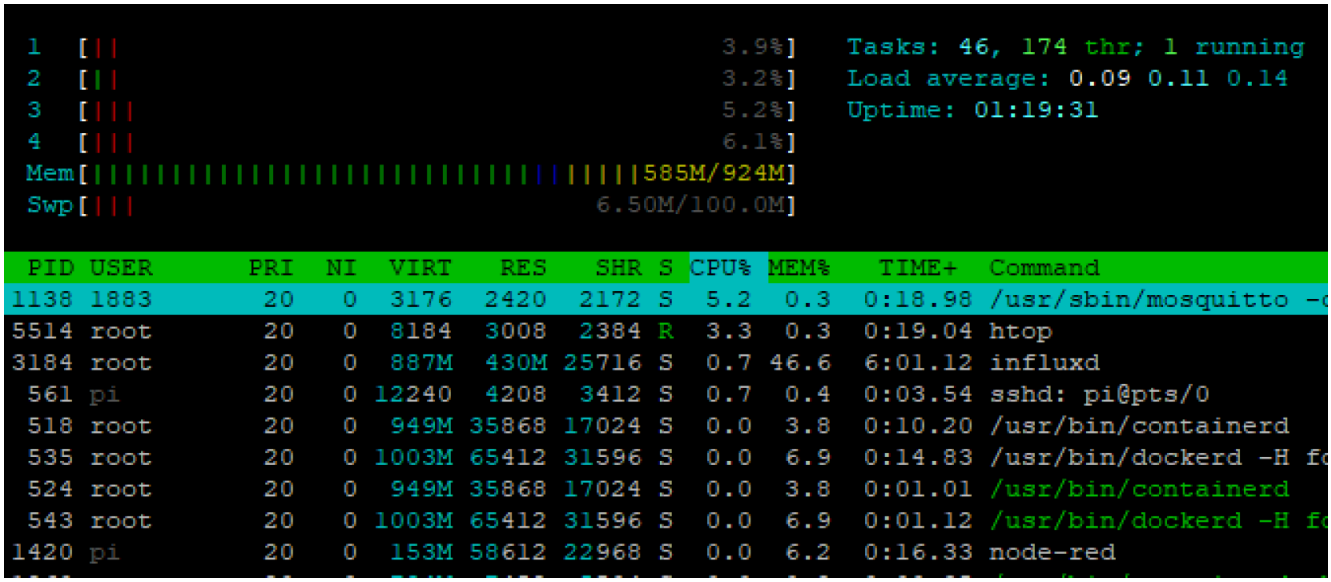


Рис. 3.43 Вплив кількох підключень mqtt на процесор

ВИСНОВКИ

Підсумки дослідження розробки системи обробки та моніторингу MQTT повідомлень на базі Node-Red, InfluxDB і Grafana можуть бути такими:

Загальний висновок: Дослідження показало, що система на базі Node-Red, InfluxDB і Grafana є ефективним та потужним інструментом для обробки та моніторингу MQTT даних. Вона дозволяє легко підключати та обробляти дані з різних джерел, зберігати їх у часовій базі даних, і візуалізувати на графіках та панелях.

Простота розробки: Використання Node-Red спрощує розробку системи завдяки візуальному інтерфейсу та готовим вузлам для MQTT, InfluxDB і Grafana. Це дозволяє розробнику швидко створити потоки обробки даних без необхідності написання великої кількості коду.

Моніторинг та візуалізація: Grafana надає багато можливостей для створення красивих та інтерактивних графіків та панелей для моніторингу MQTT даних. Це допомагає користувачам легко візуалізувати та аналізувати дані.

Відкритий код: Всі компоненти системи (Node-Red, InfluxDB і Grafana) є відкритими проектами з активними спільнотами користувачів та розробників. Це забезпечує доступ до вихідного коду та можливість налаштування системи під індивідуальні потреби.

Споживання ресурсів: Система може ефективно використовувати ресурси сервера, особливо при належній настройці та оптимізації InfluxDB. Це робить її придатною для роботи в різних виробничих середовищах.

Підтримка MQTT: Система підтримує MQTT як протокол для обміну даними, що робить її ідеальним вибором для проектів Інтернету речей (IoT), де MQTT часто використовується для збору та передачі даних.

Масштабованість: Систему можна легко масштабувати, додаваючи нові сервери або розширюючи кластери InfluxDB при зростанні обсягу MQTT даних.

Загалом, розроблена система є потужним та дієвим інструментом для обробки та моніторингу MQTT даних. Вона дозволяє легко збирати, обробляти,

зберігати та візуалізувати дані, що робить її корисною для широкого спектру застосувань, зокрема в галузі Інтернету речей та моніторингу систем.

Цей проект мав одну головну мету: створити вітчизняну платформу для контролю та моніторингу різних типів датчиків.

Для виконання цього завдання було розроблено дослідження різних платформ для встановлення через локальний сервер (наприклад, Raspberry Pi), щоб полегшити доступ із локальної мережі до візуалізації та моніторингу даних, що можна зробити з веб-браузера. .

Після цього дослідження було вирішено розробити нашу платформу на основі Node-RED, оскільки вона була дуже цікава завдяки блочному програмуванню, численним доступним плагінам і повній кастомізації.

Потім, після завершення базової інсталяції, були додані вдосконалення, такі як база даних для зберігання історичних даних, інший тип більш повного представлення, упакованого докерами, тощо.

Підсумовуючи, можна було здійснити базову, але повну інсталяцію домашньої платформи для моніторингу домашніх даних і дій.

В якості майбутнього розвитку можна реалізувати багато речей, наприклад, кілька датчиків, інтеграцію з такими платформами, як Telegram і Google Assistant, відображення показників у режимі реального часу, серед багатьох інших. Це найцікавіше у світі IoT, який постійно розвивається.

Державний університет інформаційно-комунікаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

КВАЛІФІКАЦІЙНА РОБОТА
на тему:
“Розробка системи обробки та моніторингу MQTT повідомлень на основі Node-Red, InfluxDB і Grafana”

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

Виконав: здобувач вищої освіти гр. ІСДМ-61
Кирило ВАШКУЛАТ

- ***Актуальність теми:*** Робота є актуальною у зв'язку зі зростанням інтересу до Інтернету Речей та потреби в ефективних інструментах обробки та моніторингу даних з різних джерел. Розроблена система на базі Node-Red, InfluxDB і Grafana відповідає цим вимогам, забезпечуючи можливість легкої інтеграції, візуалізації та аналізу MQTT даних, що робить її корисною для широкого спектру застосувань в галузі IoT та моніторингу систем.
- ***Об`єкт дослідження:*** система обробки та моніторингу MQTT повідомлень на базі Node-Red, InfluxDB і Grafana, яка представляє собою комплексне явище в галузі інтернету речей.
- ***Предмет дослідження:*** аспекти розробленої системи, такі як її ефективність, простота розробки, масштабованість, візуалізація даних, відкритий код тощо, що визначається як основні характеристики та функціональність.
- ***Мета дослідження:*** розробка та вивчення системи обробки та моніторингу MQTT повідомлень на базі Node-Red, InfluxDB і Grafana з метою створення ефективного інструменту для контролю та візуалізації даних в інтернеті речей
- ***Завдання дослідження:***
  - огляд Node-Red, InfluxDB і Grafana;
  - встановлення та налаштування Docker контейнера;
  - розгортання InfluxDB та Grafana.

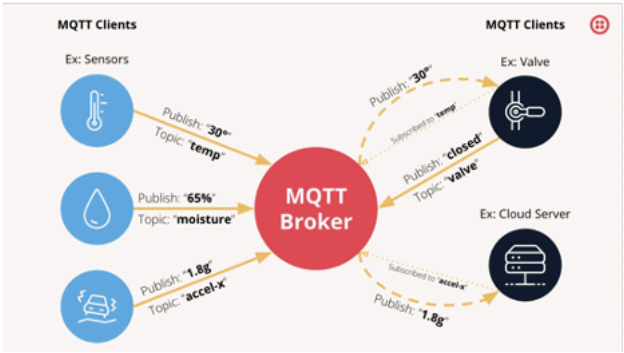


Рис. 3.1 - Принцип роботи протоколу MQTT

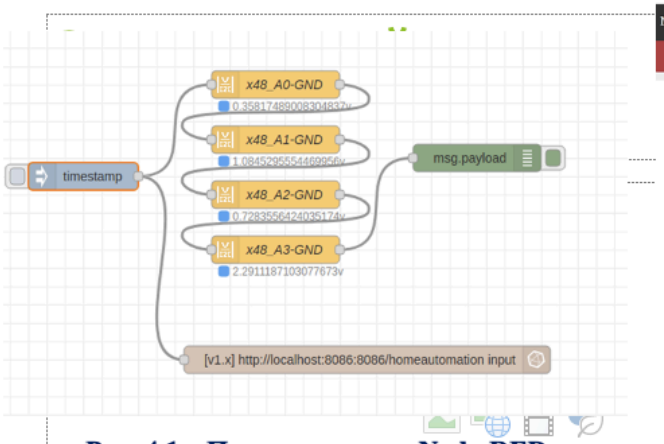


Рис. 4.1 – Приклад потоку Node-RED

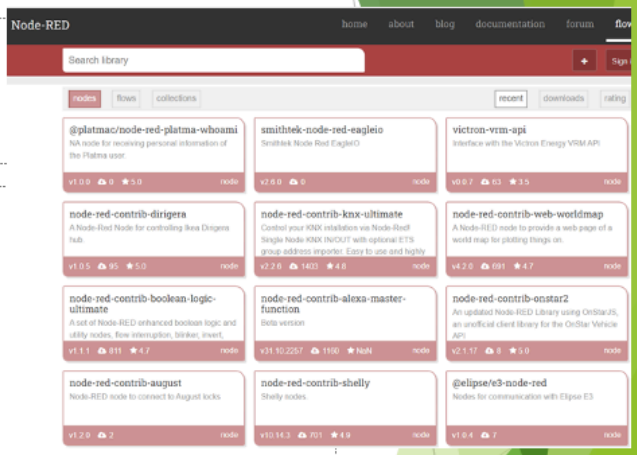


Рис.4.2 - Бібліотека вузлів Node-RED

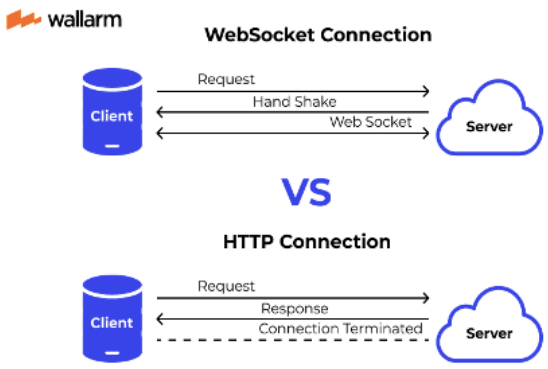


Рис.5.3 - Принципи роботи Websocket та HTTP

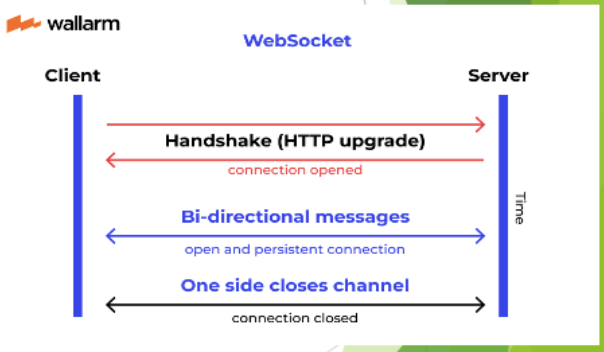
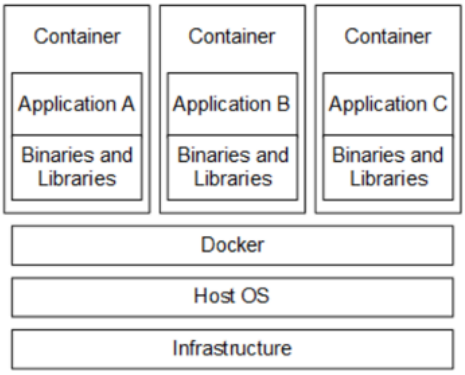


Рис.5.3 - Принцип роботи Websocket

Таблиця 6.2 – Створення і налаштування графіків та панелей для моніторингу MQTT даних в Grafana



Таблиця 6.1 – Спосіб роботи контейнерних програм

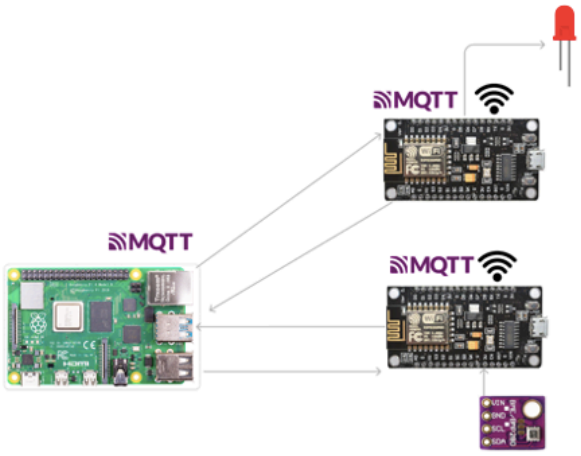


Рис.7.3 – Архітектура системи: Raspberry Pi 3B+, ESP8266 з сенсором BME280 та ESP8266 з світлодіодом для сповіщень

Встановлення та налаштування Docker контейнера

```
pi@osboxes:~ $ curl -fsSL https://get.docker.com -o get-docker.sh
pi@osboxes:~ $ sudo sh get-docker.sh
# Executing docker install script, commit: f45d7c11389849ff46a6b4d94e0dd1ffebca32c1
+ sh -c apt-get update -qq >/dev/null
```

Рис.8.1 – Запуск скрипта завантаження Docker

```
pi@raspberrypi:~ $ sudo apt-get update
Hit:1 http://raspbian.raspberrypi.org/raspbian buster InRelease
Hit:2 http://archive.raspberrypi.org/debian buster InRelease
Hit:3 https://download.docker.com/linux/raspbian buster InRelease
Reading package lists... Done
pi@raspberrypi:~ $ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  alsa-utils libssl1.1 openssh-client openssh-server openssh-sftp-server openssl raspberrypi-sys-mods rpi-eeprom ssh
9 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 4,780 kB of archives.
After this operation, 1,568 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Рис.8.1 – Оновлення пакетів linux

Налаштування вузлів Node-RED

Рис.10.1 – Інтерфейс вузла node-red-dashboard

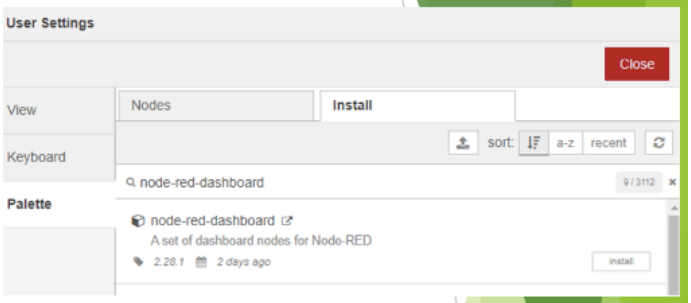


Рис.10.2 – Пошук та встановлення вузлів Node-RED

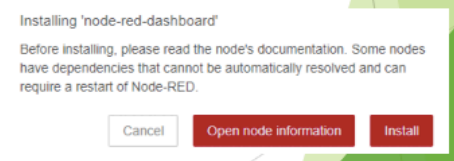


Рис.10.3 – Встановлення MQTT брокера в Node-RED

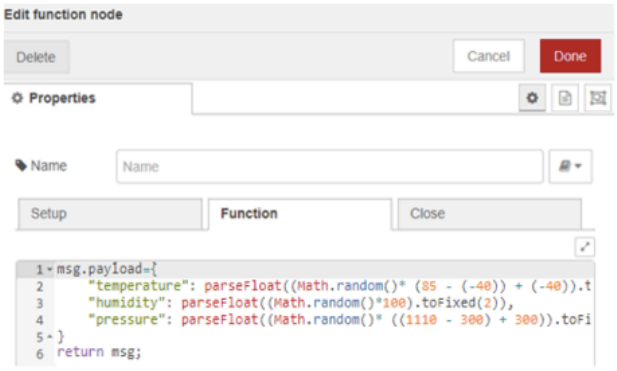


Рис.11.1 - Налаштування симуляційних даних Function node

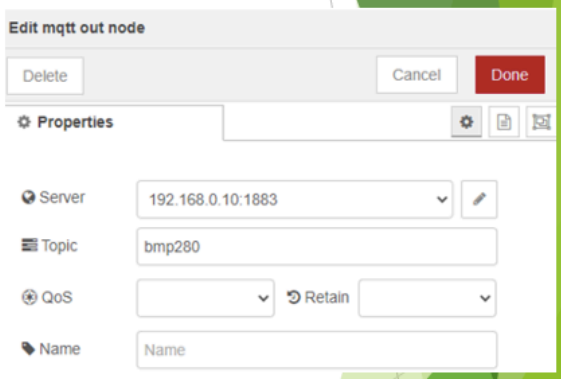


Рис.11.2 - Налаштування вузла MQTT брокера

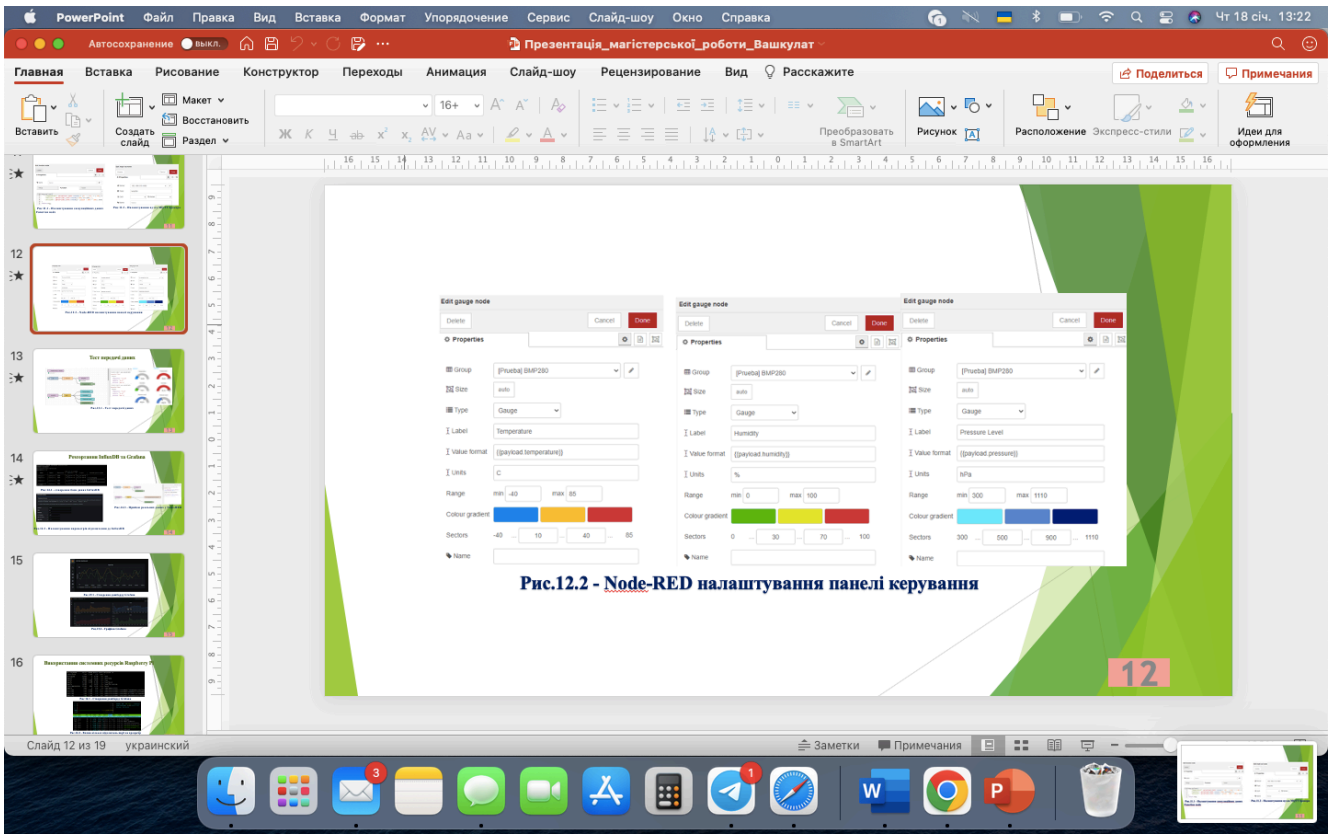


Рис.12.2 - Node-RED налаштування панелі керування

Рис.12.2 - Node-RED налаштування панелі керування

Тест передачі даних

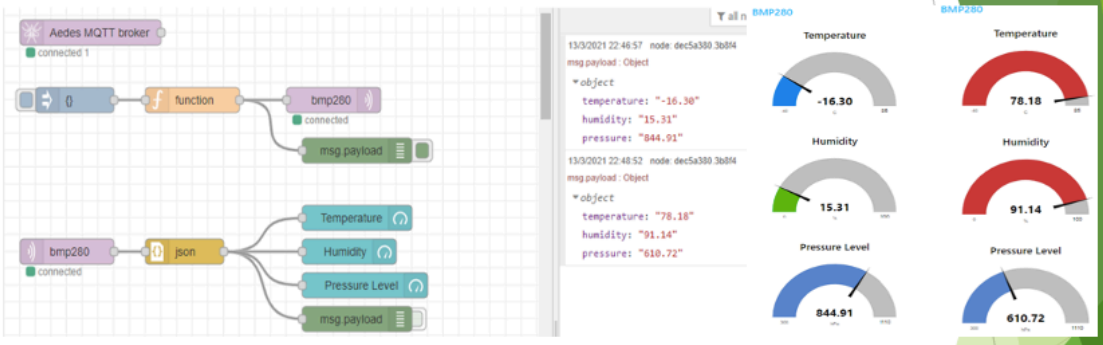


Рис.13.1 - Тест передачі даних

Розгортання InfluxDB та Grafana

```
postgres=# CREATE DATABASE thingsboard;
CREATE DATABASE
postgres=# show databases
postgres-# \l
                                  List of databases
    Name     |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-------------+----------+----------+-------------+-------------+-----------------------
 postgres    | postgres | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 |
 template0   | postgres | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 | =c/postgres          +
             |          |          |             |             | postgres=CTc/postgres
 template1   | postgres | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 | =c/postgres          +
             |          |          |             |             | postgres=CTc/postgres
 thingsboard | postgres | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 |
(4 rows)
```

Рис.14.1 - створення бази даних InfluxDB

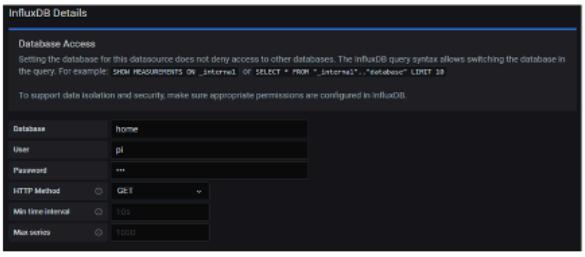


Рис.14.2 - Налаштування параметрів підключення до InfluxDB

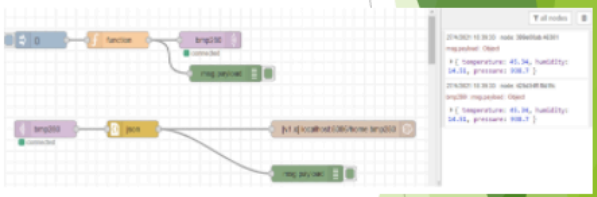


Рис.14.3 - Прийом реальних даних у Node-RED

Рис.15.1 - Створення дашборду Grafana

Рис.15.2 - Графіки Grafana

Використання системних ресурсів Raspberry Pi

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        14G  2.6G   11G  20% /
devtmpfs        430M     0  430M   0% /dev
tmpfs           463M     0  463M   0% /dev/shm
tmpfs           463M  6.5M  456M   2% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           463M     0  463M   0% /sys/fs/cgroup
/dev/mmcblk0p1  253M   48M  205M  19% /boot
tmpfs            93M     0   93M   0% /run/user/1000
overlay          14G  2.6G   11G  20% /var/lib/docker/overlay2/7f6da8ablefc729b
overlay          14G  2.6G   11G  20% /var/lib/docker/overlay2/4f6aa62583c01ca8c
overlay          14G  2.6G   11G  20% /var/lib/docker/overlay2/612c9f15d23362e81
overlay          14G  2.6G   11G  20% /var/lib/docker/overlay2/9e099dca0c875c7fe
```

Рис.16.1 - Створення дашборду Grafana

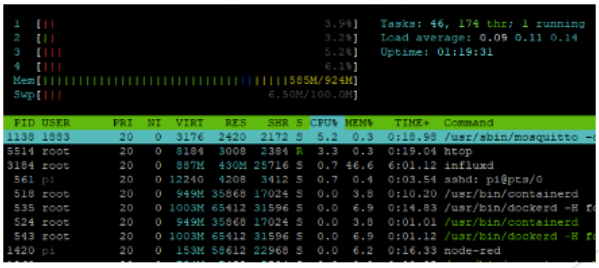


Рис.16.2 - Вплив кількох підключень mqtt на процесор

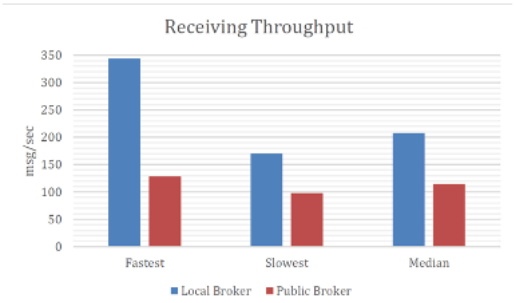


Рис.17.1 - Створення дашборду Grafana

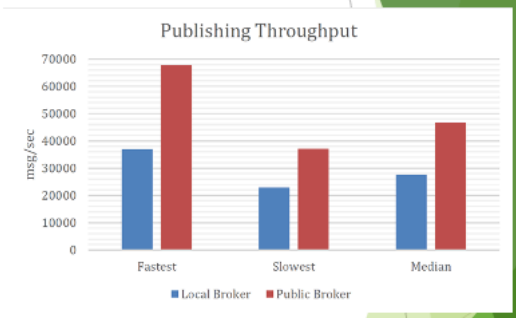


Рис.15.1 - Mqtt-stresser результати публікації повідомлень клієнтів 10, Повідомлень 100

17

ВИСНОВКИ

- Дослідження показало, що система на базі Node-Red, InfluxDB і Grafana дозволяє легко підключати та обробляти дані з різних джерел, зберігати їх у часовій базі даних, і візуалізувати на графіках та панелях.
- Використання Node-Red спрощує розробку системи завдяки візуальному інтерфейсу та готовим вузлам для MQTT, InfluxDB і Grafana, що дозволяє розробнику швидко створити потоки обробки даних без необхідності написання великої кількості коду.
- Grafana надає багато можливостей для створення красивих та інтерактивних графіків та панелей для моніторингу MQTT даних. Це допомагає користувачам легко візуалізувати та аналізувати дані.
- Систему можна легко масштабувати, додаваючи нові сервери або розширюючи кластери InfluxDB при зростанні обсягу MQTT даних.

18

18

Апробація результатів дослідження:

1. Вашкулат К.С. «Розробка архітектури системи обробки та моніторингу MQTT повідомлень». Тези доповіді на І Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу».
2. Вашкулат К.С. «Аналіз сучасних вимог до систем обробки та моніторингу mqtt повідомлень». І Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу».

ДЯКУЮ ЗА УВАГУ!