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

Інститут телекомунікаційних систем

Кафедра Телекомунікаційних систем

«На правах рукопису»
УДК 621.391.3

«До захисту допущено»
Завідувач кафедри
_____ Л.О. Уривський
«___»_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 172 Телекомунікації та радіотехніка

на тему: «Дослідження методики захисту мереж IoT від несанкціонованого доступу»

Виконав (-ла):
студент (-ка) ІІ курсу, групи ТС-81м
Радчук Олександр Володимирович _____

Керівник:
Доцент кафедри ТК, к.т.н., доцент
Міночкін Д.А. _____

Рецензент:
Доцент кафедри ІТМ, к.т.н., доцент
Правило В.В. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут телекомунікаційних систем

Кафедра Телекомунікаційних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) – 172 «Телекомунікації та радіотехніка» (172.3620.1 «Телекомунікаційні системи та мережі»)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Л.О. Уривський

«___»_____2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Радчуку Олександру Володимировичу

1. Тема дисертації «Дослідження методики захисту мереж IoT від несанкціонованого доступу», науковий керівник дисертації Міночкін Дмитро Анатолійович, кандидат технічних наук, доцент, затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом дисертації 9 грудня 2019 року.

3. Об’єкт дослідження мережі IoT.

4. Предмет дослідження можливості захисту мереж IoT засобами хмарних технологій за допомогою запропонованого рішення.

5. Перелік завдань, які потрібно розробити.

- Дослідження сучасних мереж IoT.
- Огляд наявних безпекових рішень в мережах IoT.
- Створення системи для використання SeCaaS в мережах IoT.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу
Плакат № 1 «Тема, мета, актуальність, об’єкт, предмет, проблематика, завдання дослідження»»
Плакат № 2 «Минуле, теперішнє та майбутнє IOT технологій»
Плакат № 3 «Архітектура екосистеми IOT»
Плакат № 4 «Запропоновані формати даних і протоколи їх передачі між компонентами системи IOT»

Плакат № 5 «Лямбда- архітектура»
Плакат № 6 «Висновки»

7. Орієнтовний перелік публікацій - Minochkin D. A., Radchuk A.V. SECURITY ISSUES THAT WILL DOMINATE IN IOT CLOSEST FUTURE - Institute of Telecommunication Systems, Igor Sikorsky Kyiv Polytechnic Institute, Ukraine

8. Дата видачі завдання 1 жовтня 2018 року.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Аналіз актуальності тематики та пошук наукової літератури за темою дисертації	Жовтень – Листопад 2018р.	
2	Аналіз існуючих методів побудови мереж IoT	Грудень 2018р.	
3	Аналіз існуючих методів захисту телекомунікаційних мереж від несанкціонованого доступу IoT	Лютий-Березень 2019р.	
4	Підготовка тез доповіді на конференцію «Проблеми телекомунікацій-2019»	Березень 2019р.	
5	Підготовка тез доповіді до конференції Молодих вчених	Квітень 2019р.	
6	Звіт науковому керівнику за 1й рік навчання на магістратурі.	Квітень 2019р.	
7	Формування змістової частини роботи	Вересень – Листопад 2019р.	
8	Підготовка до захисту дипломної роботи	Листопад – Грудень 2019р.	

Студент Радчук О.В.

Науковий керівник дисертації Міночкін Д.А.

РЕФЕРАТ

магістерської дипломної дисертації Радчука Олександра Володимировича
на тему: «Дослідження методики захисту мереж IoT від
несанкціонованого доступу»

Представлена робота націлена на створення екосистемного рішення для систем IoT, що конвергує їх з безпековими рішеннями SeCaaS (Security as a Serves).

В роботі запропоновано платформу для збору, зберігання та обробки великих обсягів даних. В цілях надання більшої гнучкості та додаткових можливостей для використання хмарних рішень з питань безпеки. Для створення такої платформи використовувалися сучасні бібліотеки та бази даних із відкритим вихідним кодом, а також найновіші протоколи передачі даних. Було досліджено варіанти по можливим шляхам захисту систем IoT, розписані їх недоліки та можливе майбутнє. Архітектурне рішення платформи, дає можливість для її високої масштабованість і ефективності.

Програмна частина платформи також приведена та може бути настроєна під вимоги конкретної IoT мережі. Лістинги програм приведені в додатку 1.

Загальний обсяг роботи: 75 сторінок, 15 рисунків, 1 таблиця, , 1 додаток.

Ключові слова: ІНТЕРНЕТ РЕЧЕЙ, ХМАРНА ПЛАТФОРМА, РОЗПОДІЛЕНА СИСТЕМА, ОБРОБКА ВЕЛИКИХ ОБСЯГІВ ДАНИХ, МОНІТОРИНГ МЕРЕЖІ ПРИСТРОЇВ, APACHE SPARK, APACHE KAFKA, APACHE CASSANDRA, MONGODB.

РЕФЕРАТ

магистерской дипломной диссертации Радчука Александра Владимировича на тему: «Исследование методики защиты сетей IoT от несанкционированного доступа»

Представленная работа нацелена на создание экосистемного решения для систем IoT, что конвертирует их с вопросами безопасности решениями SeCaaS (Security as a Serves).

В работе предложено платформу для сбора, хранения и обработки больших объемов данных. В целях придания большей гибкости и дополнительных возможностей для использования облачных решений по вопросам безопасности. Для создания такой платформы использовались современные библиотеки и базы данных с открытым исходным кодом, а также новейшие протоколы передачи данных. Было исследовано варианты по возможным путям защиты систем IoT, расписаны их недостатки и возможное будущее. Архитектурное решение платформы, дает возможность для ёё высокой масштабируемостью и эффективности.

Программная часть платформы также приведена и может быть настроена под требования конкретной IoT сети. Листинги программ приведены в приложении 1.

Общий объем работы: 75 страниц, 15 рисунков, 1 таблица, 1 приложение.

Ключевые слова: Интернет вещей, облачная платформа, распределенная система, обработка больших объемов данных, мониторинг сети устройств, Apache Spark, Apache Kafka, Apache Cassandra, MongoDB.

ABSTRACT

of the master's thesis of Alexandr Radchuk " The method to protection IoT networks investigation against unauthorized access "

The presented work is aimed at creating an ecosystem solution for IoT systems, which converts them with security issues using SeCaaS (Security as a Serves) solutions.

The work offers a platform for collecting, storing and processing large amounts of data. In order to give greater flexibility and additional opportunities for the use of cloud security solutions. To create such a platform, modern libraries and open source databases were used, as well as the latest data transfer protocols. The options for possible ways to protect IoT systems were investigated, their shortcomings and a possible future were outlined. The architectural solution of the platform makes it highly scalable and efficient.

The software part of the platform is also provided and can be customized to the requirements of a specific IoT network. The listings of programs are given in Appendix

1.Key words: Internet of things, cloud platform, distributed system, big data processing, device network monitoring, Apache Spark, Apache Kafka, Apache Cassandra, MongoDB.

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ

IoT – Internet of things – Інтернет речей.

M2M (Machine-to-Machine) - машино-машинна взаємодія.

SIoT (social IoT) – соціальний інтернет речей.

WOF (web of things) – глобальга мережа речей.

AI (Artificial intelligence) - Шту́чний інтеле́кт.

EiF(Elastic Intelligent Fog) – розумний еластичний туман.

PaSS(Platform as a Service) – платформа як послуга.

API(Application Programming Interface) - Прикладни́й програ́мний інтерфе́йс.

MQTT (Message Queue Telemetry Transport) - спрощений мережевий протокол, що працює на TCP / IP.

RFID(Radio Frequency IDentification, радиочастотная идентификация) – модуль радіочастотної ідентифікації.

UWB (Ultra-Wide Band) – над висока смуга.

TLS(transport layer security) - захист на транспортному рівні.

SOC(security operations center) – оперативний центр безпеки.

SQL(Structured Query Language) - мова структурованих запитів.

JSON(JavaScript Object Notation)

SeCaaS(Security as a Service) - Безпека як послуга.

QoS(quality of service) – якість послуги.

GNU(General Public License) – публічна ліцензія.

IEEE(nstitute of Electrical and Electronics Engineers) - Інститут інженерів з електротехніки та електроніки.

IETF(nternet Engineering Task Force) - відкрите міжнародне співтовариство проектувальників.

GPU(graphics processing unit) - Графічний процесор.

ВСТУП

В останні роки можемо спостерігати поширення Інтернету речей (IoT), зростає кількість вже підключених до Інтернету пристроїв, шо складає кілька мільярдів. Ця тенденція супроводжується вдосконалення систем IoT, обумовлених зростаючим попитом та галузями їх застосування, наприклад, безпілотники, роботи та автономні керовані транспортні засоби. У цій ситуації користувачі IoT стикаються з нагальними проблемами безпеки, які включають заводські вразливості та незахищеність від хакерських атак. Час вимагає нових і розумніших підходів до безпеки IoT, особливо підходів, які здатні вирішити складні та непередбачувані виклики.

Ключовою умовою для забезпечення таких підходів є розробка масштабованих інфраструктурних рішень для збору та обробки наборів даних, пов'язаних з безпекою, від IoT систем та пристроїв. Також при пошуку можливих рішень було взято до уваги можливість імплементації традиційних методів забезпечення безпеки, та їх варіації.

У цій роботі представлено детальний аналіз можливих рішень з фінальним вибором найбільш вдалого варіанту для систем IoT. Вона пропонує первинну кластеризацію IoT пристроїв, збір даних з різних елементів систем IoT, включаючи окремі пристрої та розумні об’єкти, крайові вузли, платформи IoT та цілі хмари, а також їх структуризацію та зберігання, з метою їх подальшої передачі до вендорів SeCaaS.

Масштабованість впровадженої інфраструктури випливає з інтеграції сучасних технологій для великого масштабування збору, потокової передачі та зберігання даних. Цей підхід дозволяє створити швидко та відносно легко реалізуємо систему збору даних та передання їх обраному оператору хмарних технологій.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ Й ІСНУЮЧИХ РІШЕНЬ

1.1 Предметна область і актуальність роботи

Інтернет речей (IoT) - це система взаємопов’язаних обчислювальних пристроїв, механічних і цифрових машин, предметів, тварин або людей, яким надаються унікальні ідентифікатори (UID) та можливість передавати дані по мережі без необхідності людини від людини. або взаємодія людини з комп'ютером.

Народження термін IoT отримав з подачі аналітиків корпорації Cisco, які порахували, що в період з 2008 по 2009 рік кількість пристроїв, підключених до глобальної мережі, перевищила чисельність населення Землі, тим самим «Інтернет людей» став «Інтернетом речей». Влітку 2013 року Cisco запустила лічильник підключених до Інтернету пристроїв Connections Counter.

Темп підключення фізичних пристроїв навколо нас до Інтернет швидко зростає. За даними нещодавнього Gartner Звіт, буде близько 8,4 мільярда пов'язаних речей в усьому світі в 2020 році[6]. Очікується, що ця кількість зросте 20,4 млрд до 2022 р. Використання програм IoT є зростає у всіх частинах світу. Основні країни, що рухаються сюди входять Західна Європа, Північна Америка та Китай. Кількість з'єднань машина-машина (M2M) очікується, що зросте з 5,6 мільярда в 2016 році до 27 мільярдів у 2024 році. Цей стрибок у числах сам заявляє IoT - один з головних майбутніх ринків, який міг би утворюють наріжний камінь розширюваної цифрової економіки.

Очікується, що галузь IoT зростатиме з точки зору доходу від 892 мільярди доларів у 2018 році до 4 трильйонів доларів до 2025 року. M2M-з'єднання охоплюють широкий спектр застосувань, таких як розумні міста, розумне середовище, розумні сітки, розумна роздрібна торгівля, розумне господарство тощо.

Надалі пристрої не тільки очікують бути підключеним до Інтернету та інших локальних пристроїв, але очікується також спілкування з іншими пристроями безпосередньо. Крім пристроїв чи речей пов'язане, виникає також концепція соціального IoT (SIoT). SIoT дозволить підключати різних користувачів соціальних мереж на пристрої та користувачі можуть надавати

спільний доступ до пристроїв Інтернет. При всьому цьому величезному спектрі застосувань IoT приходить питання безпеки та конфіденційності. Без надійної та сумісної роботи Екосистема IoT, нові програми IoT не можуть охопити високий попит і може втратити весь свій потенціал.[1]

Поряд із проблеми безпеки, з якими стикаються Інтернет, стільникові мережі, і WSN, IoT також має свої особливі проблеми безпеки наприклад, питання конфіденційності, проблеми аутентифікації, управління проблеми, зберігання інформації тощо.

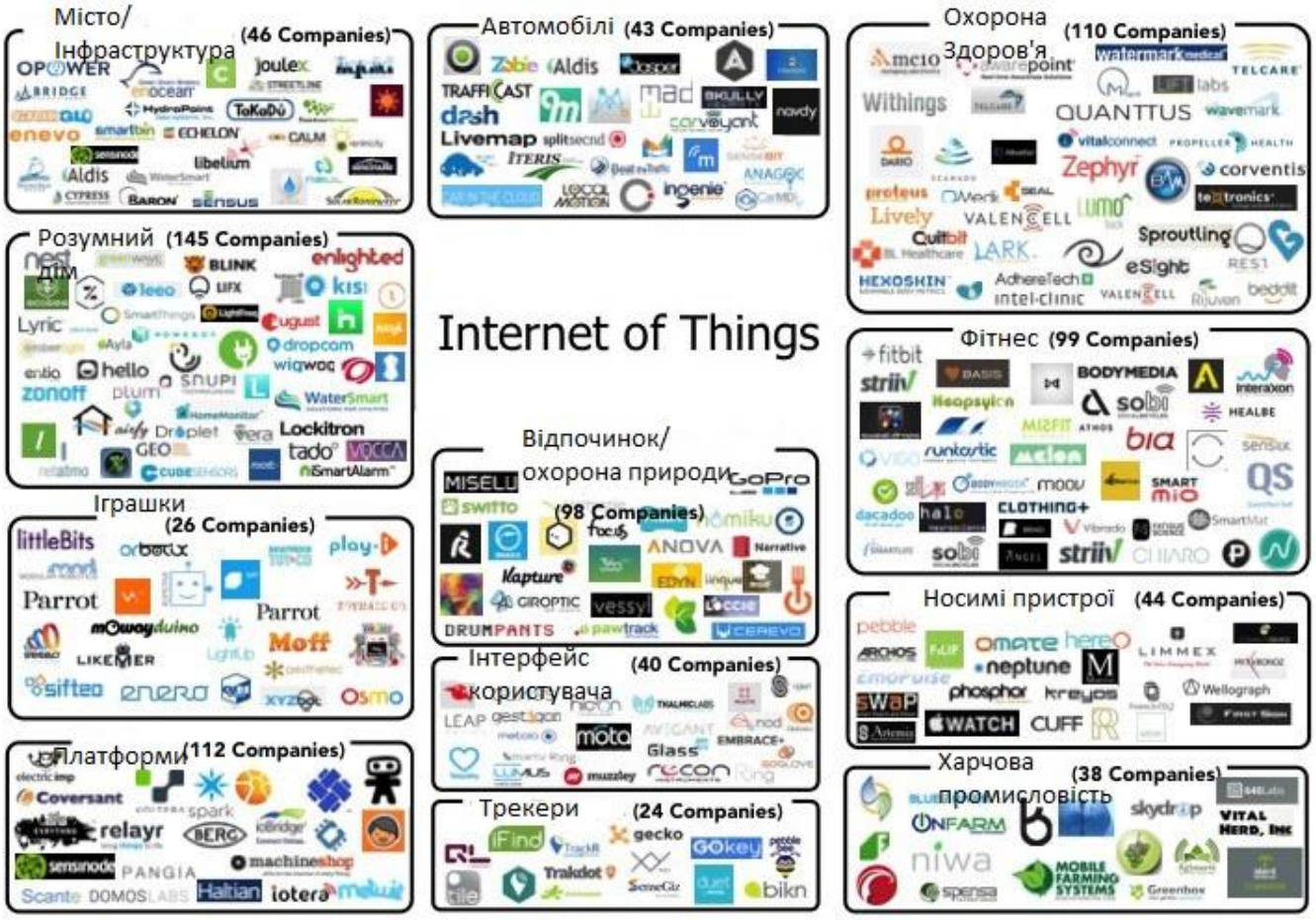

Рисунок 1.1 – Ландшафтна схема підприємств працюючих на ринку IoT. Данні компанії Venture Scanner [6]

1.2 Технічна сутність Інтернету речей

З технічної точки зору Інтернет речей не є результатом єдиної нової технології; натомість поєднання кілька додаткових технічних розробок надають можливості, які разом допомагають подолати розрив між віртуальним та

фізичним світом. До таких можливостей належать:

- Спілкування між пристроями та кооперація: Об'єкти мають можливість мережі з Інтернет-ресурсами або навіть між собою, користуватися даними та послугами та оновлювати свій стан. Бездротові технології, такі як GSM та UMTS, Wi-Fi, Bluetooth, ZigBee та різні інші стандарти бездротових мереж, які зараз розробляються, особливо ті, що стосуються Wireless Personal Area Networks (WPAN), мають першочергове значення тут.
- Адресність: в Інтернеті речей об’єкти можна розміщувати та вирішувати за допомогою відповідних служб виявлення, пошуку або іменування, а отже, допитуватися чи конфігуруватися віддалено.
- Ідентифікація: об'єкти є унікально ідентифікованими. RFID, NFC (Near Field Communication) і оптично зчитуються штрих-коди є прикладами технологій, за допомогою яких можна ідентифікувати навіть пасивні об'єкти, які не мають вбудованих енергетичних ресурсів (за допомогою «посередника», такого як зчитувач RFID або мобільний телефон Телефон). Ідентифікація дозволяє пов'язувати об'єкти з інформацією, пов'язаною з конкретним об'єктом і яка може бути отримана з сервера, за умови, що посередник підключений до мережі (див. Рисунок 1).
- Передавання даних: Об'єкти збирають інформацію про навколишнє середовище за допомогою датчиків, записують її, пересилають її або реагують безпосередньо на неї.
- Активація: Об'єкти містять виконавчі механізми для управління навколишнім середовищем (наприклад, шляхом перетворення електричних сигналів в механічний рух). Такі приводи можуть використовуватися для дистанційного керування реальними процесами через Інтернет.
- Вбудована обробка інформації: Смарт-об'єкти оснащені процесором або мікро контролером, а також ємністю пам'яті. Ці ресурси можуть використовуватися, наприклад, для обробки й інтерпретації сенсорної інформації або для того, щоб дати продуктам «пам'ять» про те, як вони використовувалися.
- Локалізація: Розумні речі знають про свій фізичний місцезнаходження або можуть бути виявлені. GPS або мережу мобільного зв'язку є придатними технологіями для досягнення цієї мети, а також вимірювання часу ультразвуку,

UWB (надширокосмугових діапазон), радіомаяків (наприклад, сусідніх базових станцій WLAN або RFID-зчитувачів з відомими координатами) і оптичних технологій.

- Інтерфейси для користувача. Смарт-об'єкти можуть спілкуватися з людьми відповідним чином (прямо або побічно, наприклад, через смартфон). Тут важливі інноваційні парадигми взаємодії, такі як відчутні призначені для користувача інтерфейси, гнучкі дисплеї на основі полімерів і методи розпізнавання голосу, віку або жестів.

Більшості конкретних програм потрібно тільки підмножина цих можливостей, особливо тому, що реалізація всіх їх часто дорога і вимагає значних технічних зусиль. Наприклад, додатки логістики в даний час концентруються на приблизною локалізації (тобто положенні останньої точки зчитування) і відносно недорогий ідентифікації об'єктів з використанням RFID або штрих-кодів. Дані датчиків (наприклад, для контролю холодних ланцюгів) або вбудованих процесорів обмежуються тими додатками логістики, де така інформація важлива, наприклад, транспортування вакцин з контролем температури.[7]

Передвісники спілкування з предметами повсякденного вжитку вже очевидні, особливо в зв'язку з RFID - наприклад, передача ключів на короткі відстані з дверима готельних номерів або скі-паси, які говорять на підйомниках турнікетів. Більш футуристичні сценарії включають в себе стіл для розумних гральних карт, де хід гри контролюється з використанням RFID-обладнаних гральних карт. Однак всі ці додатки поки задіють виділені системи в локальному розгортанні; ми не говоримо про «Інтернеті» в значенні відкритою, масштабованої і стандартизованої системи.

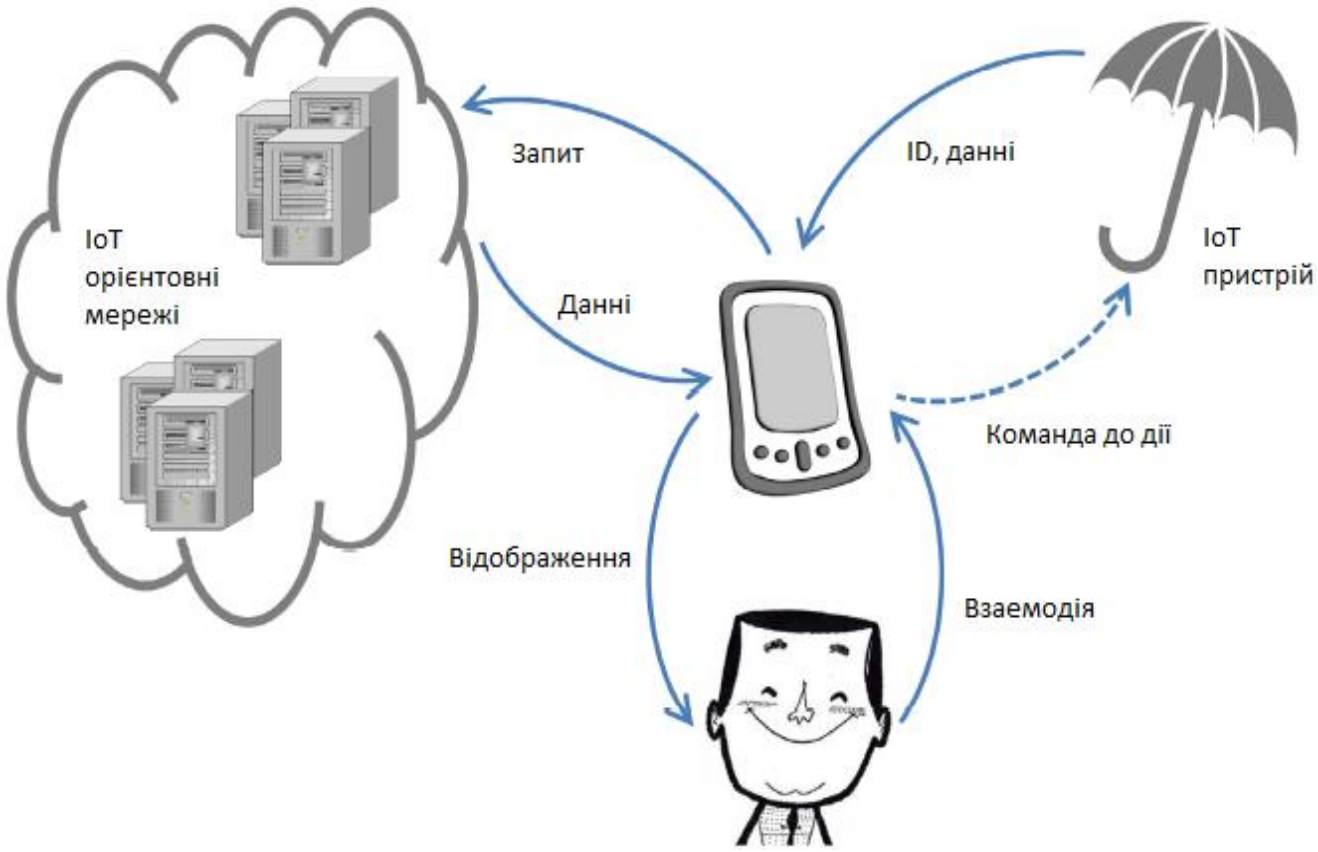

Рисунок 1.2 – Смартфон як посередник між людьми, речами та спеціалізованими мережами [7]

1.3 Джерело вразливостей систем IoT

Основна проблема полягає в тому, що, оскільки ідея мережевих пристроїв та інших об'єктів відносно нова, безпека за звичай не враховувалася при розробці продукту. IoT продукти часто продаються зі старими і непатечтними операційними системами і програмним забезпеченням. Крім того, покупці часто не можуть змінити паролі за замовчуванням на інтелектуальних пристроях - або, якщо вони все ж змінюють їх, не можуть підібрати досить надійні паролі. Для підвищення безпеки пристрій IoT, яке повинно бути безпосередньо доступно через Інтернет, має бути сегментований в свою власну мережу і мати обмежений доступ до мережі. Потім слід відстежувати сегмент мережі, щоб виявити потенційний аномальний трафік, і в разі виникнення проблеми слід вжити заходів.

Зазвичай проблеми погіршуються тим що класичні варіанти вирішення не можуть бути імплементовані для своєчасного закриття дір в безпеці.

Традиційні методи захисту SOC центри не є ефективним рішенням через неоднорідний характер пристроїв. Популярність IoT зумовлена тим що технологія дозволяє зменшити витрати на автоматизацію та моніторинг, але захист такого обладнання не є можливим. Так як в SOC звикли до захисту не великого, але дуже потужного обладнання. Таким чином загрози локалізовані, що дозволяє концентрувати увагу таким центрам безпеки на відносно не великій кількості обладнання. Що складно уявити в розрізнених мережах IoT де велика кількість пристроїв можуть одночасно бути атакованими, що зупинить підприємство.

У даному розділі представлена таблиця із висновками по основним напрямкам. Дані є консенсусом експертів що проводили аналітику підприємств з високим рівнем використанням технології IoT.

Таблиця 1. Аналітика проблем в IoT безпеці по рокам[1]

Рік	Автор	Проблеми	Рішення
2010	H. Takabi	• Аспекти що поглиблюють проблеми безпеки і конфіденційності в хмарних обчислень.	Дослідження перешкод та рішення для створення надійного середовища хмарних обчислень.
2011	S.Subashini & V. Kavitha	• Наскільки безпечне середовище хмарних обчислень. • Корпоративні клієнти як і раніше неохоче розміщують свій бізнес в хмарі. • Безпека є однією з основних проблем, яка стримує зростання хмарних обчислень, і ускладнення, пов'язані з конфіденційністю даних і захистом даних, продовжують переслідувати ринок.	• Нова модель, націлена на поліпшення характеристик існуючої моделі, не повинна ризикувати або загрожувати іншим важливих функцій поточної моделі. • Користувачі хмарних сервісів повинні бути пильні в розумінні ризиків витоку даних в цьому новому середовищі. • Різні проблеми безпеки, що виникають у зв'язку з характером моделей надання послуг в хмарі обчислювальна система.
2012	J. Gubbi	•Підтримувана недавньої адаптацією різних бездротових технологій, IoT вийшла зі свого дитинства і є наступною революційною технологією в перетворенні Інтернету в повністю інтегрований Інтернет майбутнього. Потреба в даних за запитом з використанням складних інтуїтивно зрозумілих запитів значно зростає.	• Хмара-орієнтоване бачення всесвітнього впровадження Інтернету речей. • Хмарна реалізація з використанням Aneka , яка заснована на взаємодії приватних і публічних хмар. • Розширення необхідності конвергенції WSN, Інтернету і розподілених обчислень, спрямованих а співтовариство технологічних досліджень.

Продовження таблиці

2013	G. Suciu	• Хмарні обчислення та Інтернет речей (IoT) - дві найбільш популярні парадигми ICT. • За останні кілька років зближення між хмарними обчисленнями і IoT стало темою з великою кількістю невирішених питань.	• Нова платформа для використання можливостей хмарних обчислень для надання і підтримки повсюдного підключення і додатків і сервісів в реальному часі для потреб розумних міст. • Платформа для даних, одержуваних з сильно розподілених, різнорідних, децентралізованих, реальних і віртуальних пристроїв цим можна автоматично управляти, аналізувати і контролювати розподілені хмарні сервіси.
2013	J. Zhou	• Користувач з новим засобом зв'язку зі світом Інтернету через повсюдні об'єктно-орієнтовані мережі, представлені Інтернетом речей. • Хмарні обчислення забезпечують зручний, за запитом і масштабований мережевий доступ до спільного пулу настроюються обчислювальних ресурсів.	• Загальний підхід до інтеграції Інтернету речей (IoT) і хмарних обчислень під назвою архітектури CloudThings . • Сценарій розумного будинку з підтримкою IoT для аналізу вимог до додатків IoT .
2013	M. Soliman	• Розумний будинок зводить до мінімуму втручання користувача в моніторинг домашніх налаштувань і управління побутовою технікою.	• Підходи до розвитку розумного будинку та впровадження найкращих практик • додатки шляхом інтеграції Інтернету речей (IoT) з веб-сервісами та хмарними обчисленнями.

Продовження таблиці

2014	M. Aazam	• Все буде підключено до Інтернету, і його дані будуть використовуватися для різних потенційно небезпечних цілей. • Інтернет речей (IoT) стає настільки поширеним, що стає важливим інтегрувати його з хмарними обчисленнями.	• Інтеграція IoT і хмарних обчислень не проста і має деякі ключові проблеми. Ці ключові питання разом з їх потенційними рішеннями були виділені в основні напрямки розвитку.
2014	M. Aazam	• Інтеграція Інтернету речей з хмарними обчисленнями набуває все більшого значення у зв'язку з тим, як розвиваються тенденції в світі хмарних обчислень. • Інтернет речей (IoT) стає настільки поширеним, що стає важливим інтегрувати його з хмарними обчисленнями.	• Інтеграція IoT з хмарними обчисленнями, вимагає більш інтелектуального шлюзу для виконання складних завдань і попередньої обробки, на які не здатні сенсори і легкі IoT . • Орієнтований на деякі з ключових проблем, пов'язаних з CoT і пропозицією комунікації на основі інтелектуального шлюзу.
2014	F. Tao	• Інтернет речей (IoT) і хмарні обчислення (CC) широко та в поспіху застосовувалися у багатьох областях, оскільки вони можуть надати новий метод інтелектуального сприйняття і з'єднання від M2M, а також використання і ефективного використання на вимогу. обмін на ресурси, відповідно.	• Пропонується система слуговування хмарного виробництва (CMfg) на основі CC і IoT і її архітектура. • У переваги, проблеми та майбутні роботи по застосуванню і реалізації CCIoT-CMfg є обговорили.

Продовження таблиці

2015	A. Botta	• Хмарні обчислення та Інтернет речей (IoT) - дві абсолютно різні технології, які вже є частиною сучасної високо технологічної галузі. Висока інтеграція підприємств все юільша завязана на інтренеті речей.	• Інтеграція Cloud і IoT , яка називають новою парадигмою CloudIoT . • Нова парадигма CloudIoT , яка включає в себе абсолютно нові додатки, проблеми і проблеми їх досліджень.
2015	J. A. Guerrero Ibanez	• Ефективність транспортних систем має вирішальне значення для індивідуальної мобільності, торгівлі та економічного зростання всіх країн. Вкрай важливо підвищити безпеку і ефективність транзакцій.	• Інтеграційні проблеми IoT і CC, які необхідно вирішити, щоб інтелектуальна система транспортування могла вирішувати проблеми, що стоять перед цим сектором.
2016	M. Diaz	• Інтернет речей включає в себе безліч взаємопов'язаних технологій, таких як RFID і WSAN, для обміну інформацією. Обмеження, пов'язані пристроїв в IoT вимагають в технології, такі як хмарні обчислення, щоб доповнити цю область.	• Огляд компонентів інтеграції: хмарні платформи, хмарні інфраструктури і IoT Middleware .
2016	M. Aazam	• Стає дуже важко керувати невеликими датчиками з обмеженням потужності і іншими пристроями, що генерують дані. • Згенеровані дані повинні управлятися відповідно до їх вимог, щоб створювати більш вдалі послуги з меншою кількістю потенційних дірок.	• Інтеграція IoT з хмарними обчисленнями стає дуже важливою - Cloud of Things . • CoT надають кошти для обробки зростаючих даних і інших ресурсів базових IoT і WSN.

Продовження таблиці

2016	M. lsmirat	• Велика революція в інформаційних технологіях, яка використовується для створення нової інфраструктури в хмарних технологіях, що не націлена на розподілені мережі, для більшої інтеграції з швидко ростучим ринком . • Сегментація мереж є обов'язковим етапом у багатьох процедурах аналітики, заснованих на обробці великого на неупорядкованого потоку даних. Часто в цілі сегментації не входять питання безпеки.	• Алгоритми сегментації на основі нечітких структур забезпечують точність сегментації. • Прискорте час виконання алгоритмів Fuzzy C- Means , використовуючи можливості Graphics Process Unit (GPU).
2016	B. B. Gupta	• Розпізнавання облич з відео привернуло увагу завдяки своїй популярності і простоті використання з системами безпеки, заснованими на системах спостереження. Потужна система розгорнута в Китаї. • Автоматизована система розпізнавання осіб на основі відео забезпечує величезний перелік завдань, так як необхідно виконувати перевірку особи при різних умовах. Перенесення все більше можливостей з пристроїв до централізованих серверів.	• Виконайте огляд основних методів, використовуваних для таких методів і виявлення нових тенденцій досліджень в цій області. • Узагальнити деякі добре відомі методи розпізнавання осіб в відео послідовність для застосування в біометричної безпеки і перерахуйте виникають тенденції.

Продовження таблиці

2016	Z. Zhang	• Питання соціального захисту і довіри стають все більш серйозними. Особливо на фоні проникнення IoT в велику аудиторію масового покупця. • Відсутність вивчення ефективних і дієвих оцінок і вимірювань для забезпечення безпеки і надійності різних інструментів, платформ і додатків соціальних мереж.	• Огляд сучасного стану безпеки і надійності мереж соціальних мереж, особливо в зв'язку зі зростаючою витонченістю і різноманітністю атак зі сторони зловмисників та компаній конкурентів. • Виділення нового напрямоку в оцінці та вимірюванні цих фундаментальних і базових платформ, для більш детального підстроювання систем майбутнього захисту. • Буде запропоновано нову ієрархічну архітектуру для оцінки пристроїв розташованих на виділеній території, засновану на теорії сигналізації і хмарних обчисленнях з урахуванням всіх особливостей.

1.4 Висновки до розділу 1

Інтернет речей це технологія яка розвивається дуже швидкими темпами, однією з причин є її властивість до зменшення витрат при виробництві. Таким чином IoT є однією із стовпів для наступного технологічного укладу, вона дозволяє впроваджувати на користувацький ринок технології та електроніку яка раніше не була доступною для широких мас.

Значно зросло використання IoT пристроїв за рахунок розумних міст та підсистем вбудованих в цю концепцію. Також частиною розумного міста стають всі ті носимі пристрої та телефони що є у жителів.

Велика кількість користувацьких IoT пристроїв створила нові досі не

вивчені ринки. Наразі мається тільки поверхнева аналітика, з якої можна бачити що виробники таких пристроїв не думають про безпеку взагалі. Більшість пристроїв навіть не передбачують можливості підвищення безпеки.

РОЗДІЛ 2. ОПИС ПРОБЛЕМАТИКИ ТА МОЖЛИВИХ ШЛЯХІВ ВИРІШЕННЯ ПИТАНЬ БЕЗПЕКИ ДЛЯ СИСТЕМ IOT

2.1 Реалії безпеки в IoT

Опишемо основні парадигми для забезпечення безпеки. На даний момент не має єдиного консенсусу серед інженерної спільноти (IEEE, IETF) щодо конкретних моделей організації IoT мереж. Також великою проблемою є різнорідність IoT пристроїв та їх різні технічні можливості.

Данну нішу зараз активно займають компанії технічні гіганти та нав’язують своє бачення теперішнього та майбутнього. За рахунок чого такі рішення будуть дуже швидко застарівати та заради своїх комерційних інтересів.

Ось наприклад бачення ситуації від інженерів бюро, яке розробляє архітектурні рішення для Huawei.

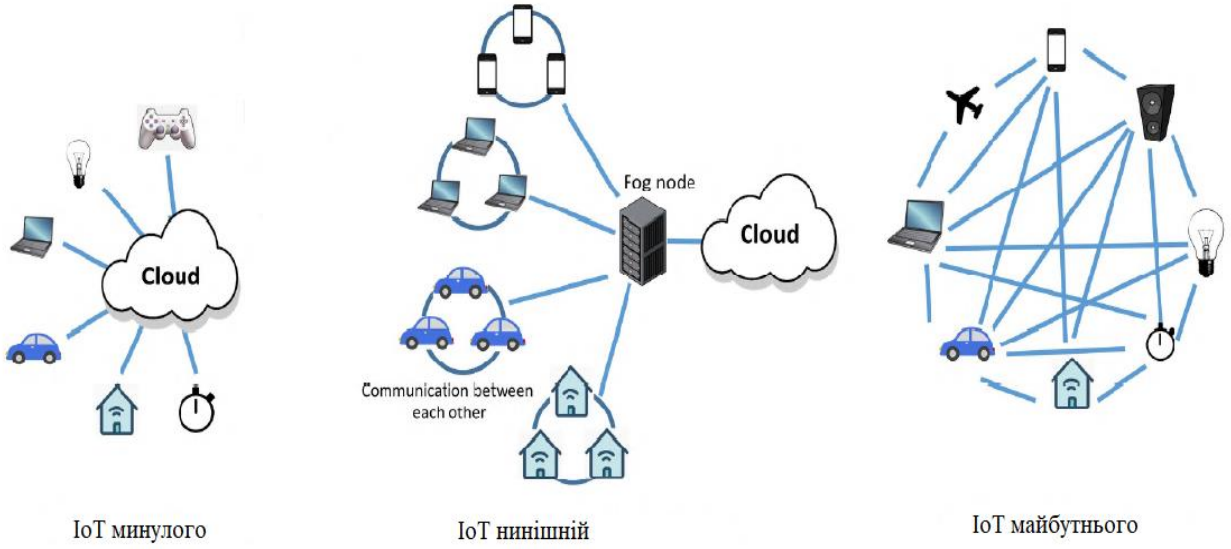

Рисунок 2.1 – Теперішнє, минуле та майбутнє IoT [4]

Даний рисунок пояснює я повинні виглядати відносини розумних речей. Можемо сміливо стверджувати, що віднесення хмарних послуг до минулого є поспішним та перед взятим рішенням. Ця технологія протримається на ринку досить довго через свої високі можливості по зменшенню витрат. Для підприємств всіх розмірів та масштабів.

Найбільш раціональним поясненням є бажання перейти до парадигми туманних обчислень. Попри затвердження щодо актуальності цього варіанту

рішення мною комерційних прикладів використання, знайдено не було.

Попри величезні обсяги наукових статей реалізація сценарію залишаться “туманною”.

Версія майбутнього інтернету речей майбутнього виглядає дуже привабливо так як не має в собі ніяких посередників у вигляді туману чи хмарних сервісів. На разі є дуже мало варіантів, в основному вони всі пов’язані з блокчейном.

2.2 Взаємопов’язані речі

Сьогодні все більше і більше пристроїв працюють навколо нас. Традиційні схеми зв'язку використовують традиційні протоколи, програмне забезпечення і призначені для користувача інтерфейси, забезпечуючи взаємодію пристроїв. Користувачі хотіли б легко отримати доступ до загальнодоступних пристроїв незалежно від вибору їх реалізації. В літературі це глобальне з'єднання пристроїв називається Інтернетом речей (IoT). Це не відноситься ні до якої технології, ні до будь-якої мережевій структурі, а відноситься тільки до ідеї взаємопов'язаних об'єктів, а також до того, як ми з'єднуємо комп'ютери з Інтернетом. В даний час все більше і більше послуг надаються в Інтернеті з використанням веб-додатків. Запропоновано використовувати цю саму модель для оточуючих пристроїв. Це називається Web of Things (WoT) ідеєю доступу до оточуючих пристроїв через веб-додатки. Ця концепція забезпечує більшу доступність для пристроїв і спрощує їх програмування. Відкриваючи нові перспективи для веб-додатків. Мережа речей вимагає вбудовування веб-серверів в невеликі пристрої. На перший погляд, реалізація HTTP / TCP / IP-стека на пристроях з декількома сотнями байтів ОЗУ і декількома кілобайтами EEPROM здається невідповідною. Вважається, що керована подіями архітектура дозволяє реалізовувати надзвичайно легкі веб-сервери з ефективною міжрівневою оптимізацією. Розробляється нова керована подіями архітектура для вбудованих веб-серверів в та доводиться можливість реалізації Web of Things через прототипування. Порівняно переваги наших пропозицій, порівнюючи наш прототип з сучасними вбудованими веб-серверами.

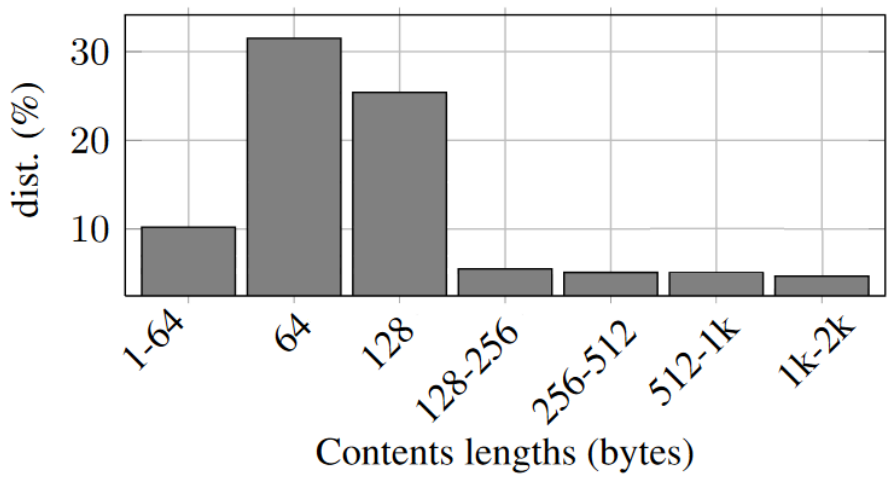

Рисунок 2.2 – Відносне розділення IoT пристроїв за їх пороговими можливостями до використання технологій блокчейну [3]

Ключові проблеми для Blockchain і IoT:

– Сумісність. Технології IoT і блокчейн з'єднують кілька пристроїв, що працюють на декількох платформах. Пристрої, які взаємодіють один з одним, можуть зіткнутися з проблемою сумісності один з одним. Для того щоб ці технології працювали ефективно, нам потрібна спільна платформа для всіх пристроїв і вбудованих технологій.

– Технічні. Основними технічними проблемами технології блокчейн і IoT є масштабованість, безпека і вимоги до сховища. Зауважимо питання масштабованості, це означає, що здатність обробляти транзакції в блокчейну обмежена. У разі фінансових транзакцій відбувається кілька тисяч транзакцій в секунду. Це означає, що в блокчейні у нас є деякі обмеження безпеки, масштабованості і ємності сховища.

2.3 Без серверні рішення або Fog

У той час як хмарні обчислення заклали основу для надання обчислювальних ресурсів кінцевим користувачам і постачальникам послуг Інтернету речей (IoT), туманні обчислення роблять свої перші кроки до того, щоб зробити хмари IoT більш еластичними («Еластичний IoT»), використовуючи можливості обчислень в рівень хмар і краї. Проте, нинішня парадигма туманних обчислень все ще стикається з проблемами, особливо коли мамо справу з

ситуаціями, які вимагають розуміння контексту і автономного прийняття рішень в умовах реального часу. Технології штучного інтелекту (II), широко використовувані в промисловості, обіцяють обчислення туману, надаючи розподілені послуги AI в вузлах туману. IoT, туманні обчислення і AI - три найбільш важливі технології для управління обчислювальної екосистемою наступного покоління. кожна технологія був розроблений незалежно.

IoT в даний час переглядає наше повсякденне життя, надаючи доступ практично до кожного фізичного об'єкту навколо світу. Ці речі можуть доставляти інформацію і надавати доступ до цих фізичним об'єктам в хмарі. Однак багато платформи IoT, що підтримують основні функції, не задовольняють вимоги багатьох галузей і областей, які шукаємо розумнішими, швидше і надійніше IoT послуги. Багато виробників і сервіс IoT провайдери приділяють більше уваги туманним обчислень через його переваг, тобто його більшої обчислювальної влада, знання і аналітика розташовані якомога ближче до користувачів. Туманні обчислення зазвичай віртуалізують мережеві ресурси і розміщують їх в туманних вузлах. Проте, при використанні IoT мережеві ресурси і платформи, послуги і знання IoT повинні розташовуватися у вузлах туману. Технології штучного інтелекту швидко розвиваються і останнім часом прискорилися таким чином, чого не можна було очікувати кілька років тому. Наприклад, технології глибокого навчання бере на себе задачу обробки неструктурованих даних, таких як відео або аудіо, в руки висококваліфікованих експертів і дозволяє більшій кількості програмістів використовувати когнітивні функції, що надаються послуги.

Однак ці технології корисні тільки в певних областях. Зусилля і дослідження, спрямовані на інтеграцію цих ключових технологій, в даний час все ще знаходяться на ранніх стадіях. Пропонутся концепція Elastic Intelligent Fog (EiF)[5], поліпшену платформу рівня послуг IoT для вузлів туману з розширеними функціями, такими як включення семантики, віртуалізація загальних сервісних функцій IoT і включення машинного навчання. Ця розширена платформа може динамічно створюватися в вузлах туману і фільтрувати непотрібні дані для швидкого прийняття рішень. Крім того, оскільки вузол туману вбудований в полегшений механізм штучного інтелекту, який може

отримувати контекстну і оперативну інформацію за допомогою управління датчиками і суміжними інтелектуальними вузлами туману, багато користувачів IoT-послуг з доданою вартістю можуть бути представлені користувачам, наприклад інтелектуальна служба моніторингу потоку трафіку.

2.3.1 Архітектура Fog

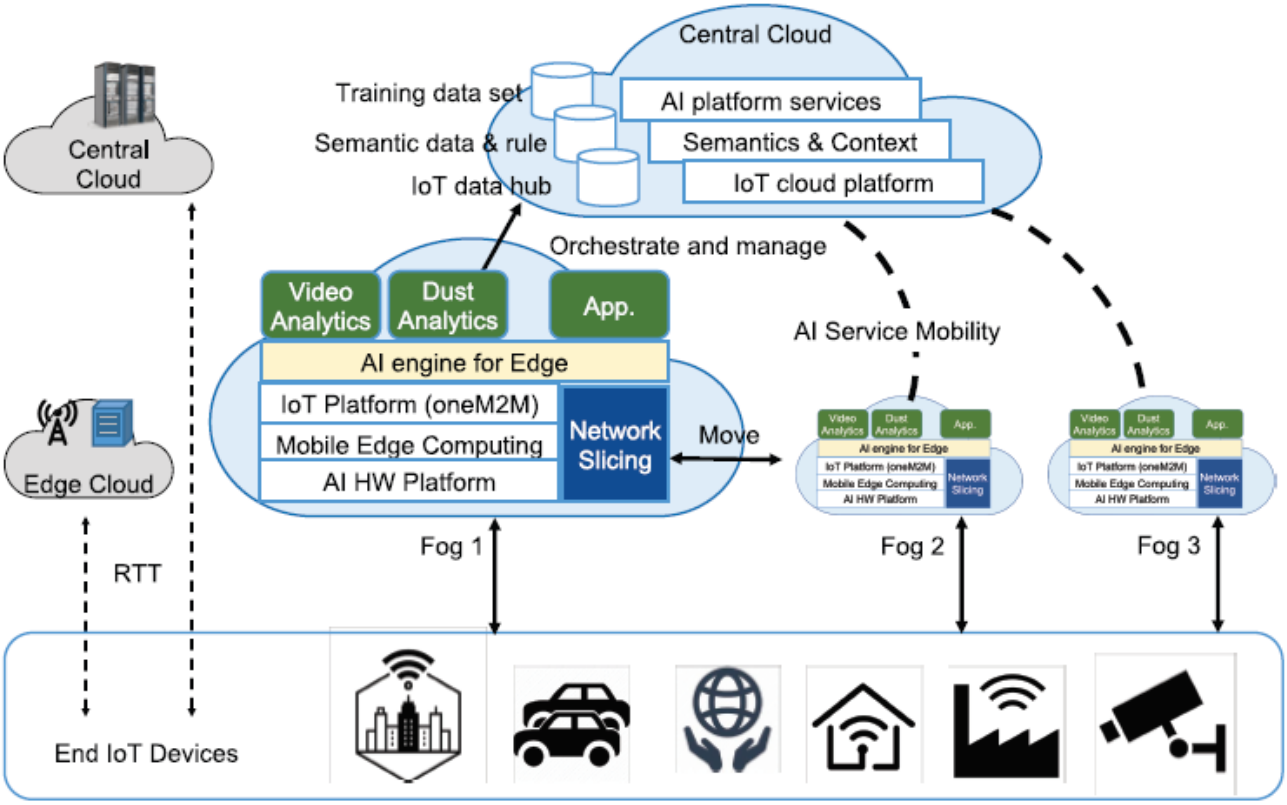

Рисунок 2.3 – Концептуальний погляд на Fog в IoT[5]

Традиційні хмарні платформи і платформи IoT не призначені для задоволення останніх виникаючих потреб в інтелектуальних додатках реального часу. Проте, Fog розширює існуючі хмарні і IoT-платформи за допомогою інфраструктури, яка дозволяє динамічно розширювати інтелектуальні функції, керовані IoT, з можливостями AI для інтелектуальних функцій.

У цьому розділі представлений огляд архітектури Fog. Основна мета Fog - надавати транскордонні послуги штучного інтелекту з використанням високо розподілених, надійних і точних систем IoT на основі туману. На малюнку 2.3 показаний концептуальний вигляд архітектури Fog. Традиційні хмарні і IoT-рішення не призначені для задоволення останніх виникаючих потреб в

інтелектуальних додатках реального часу. Проте, Fog розширює існуючі хмарні і IoT-платформи за допомогою інфраструктури, яка дозволяє динамічно розширювати інтелектуальні речі, керовані IoT, з можливостями AI для інтелектуальних речей. Сучасні методології програмного забезпечення, такі як віртуалізація і складання програм, заснованих на наміри, можуть допомогти об'єднати II і IoT в єдину систему. Центральне хмара призначене для розміщення різних функцій, які забезпечують інтелектуальні послуги AI з використанням управління даними IoT і семантичних технологій. Це центральне хмара також надає інтерфейси прикладного програмування (API), які використовуються додатками AI для використання визначених компонентів AI і IoT, що надаються рівнем «платформа як послуга» (PaaS) з будинком в різних концентраторах даних. Сумісні мережеві хмар та туману, реалізують функції IoT і функції AI для надання необхідних послуг в місцях, які знаходяться близько до користувачів. Оскільки такий спосіб взаємодії для сервісів вимагає зворотного зв'язку і обробки в реальному часі, воні мають функцію управління і реалізації необхідних віртуальних функцій в режимі реального часу. Обидва рівня обмінюються даними через стандартизований інтерфейс для синхронізації служб і даних, управління віртуальними ресурсами, забезпечення безпеки і довіри. Механізми штучного інтелекту, такі як розпізнавання осіб, виявлення аномалій і аналіз ситуації в реальному часі, інтегровані в процес аналізу потоку даних.

2.3.2 Затримки при використанні Fog

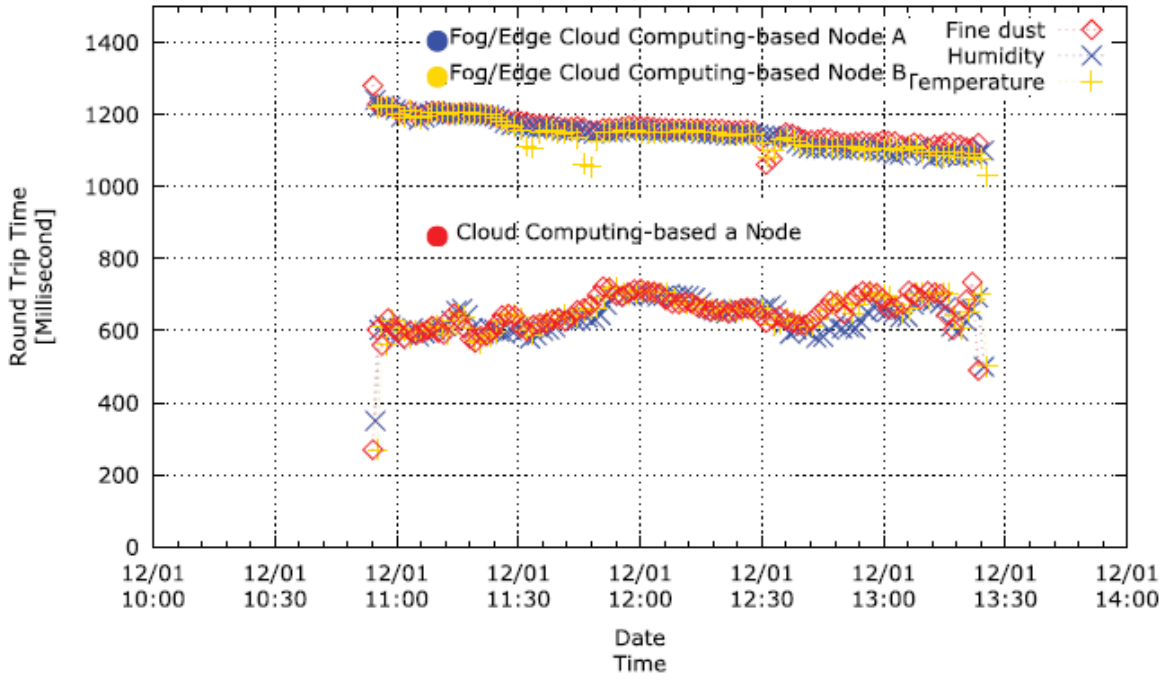

Рисунок 2.3.2 – Затримка в Fog порівняно з чистими хмарам[5]

У цьому розділі представлені результати моделювання дослідницьким центром[3], що стосуються продуктивності Fog. Використовувався сценарій з реальними умовами та реальним набором даних IoT, зібраним з Santander Smart Місто в Іспанії і набір даних портретного зображення від Google. Була продемонстрована реалізація Fog, вимірявши час кругового обходу (RTT) обміну даними IoT і результатів машинного навчання. Тут не розглянуто перевантаження в використовуваних посиланнях між об'єктами. Ми порівнюємо Fog з випадком, коли вся обробка виконувалася на центральному хмарному сервері. На рисунку 2.3.3 показано, як дані датчика IoT (тобто температура, вологість і концентрація дрібного пилу) можуть обмінюватися і оброблятися як в Fog, так і в середовищі центрального хмари. Центральний хмарний сервер отримує всю інформацію виміряних даних безпосередньо від датчиків. У разі Fog Fog-1 і Fog-2 обмінюються інформацією виміряних даних один з одним, так що навіть якщо один з них стикається з проблемою, служба може працювати через інші вузли. Цей результат ясно показує, що Fog привносить більш високу затримку даних.

2.4 Сучасні хмарні сервіси та їх виконання

2.4.1 Архітектурні та технічні особливості сучасних хмарних технологій

На даний час користувачі можуть вибирати між декількома платформами, що працюють в цій галузі. До таких платформ належать: Amazon Web Services IoT Platform, Google Cloud IoT Platform, ThingWorx, Oracle IoT, IBM IoT Platform.

Як показано на рис. 2.4, архітектура IoT AWS складається з чотирьох основних компонентів: шлюз пристроїв, механізм правил, реєстр і тіні пристроїв.

Шлюз пристроїв виступає в якості посередника між підключеними пристроями і хмарними службами, що дозволяє цим пристроям спілкуватися і взаємодіяти по протоколу MQTT. Незважаючи на те, що це старий протокол, в порівнянні з іншими протоколами IoT, Amazon використовує MQTT через декілька функцій; властивість відмовостійкості, чудове для переривчастого підключення, займає невелику кількість ресурсів з точки зору передачі, необхідного в пам'яті пристрою, дуже ефективне з точки зору вимог до пропускної здатності мережі і залежне від моделі програмування «публікація / підписка» для забезпечення зв'язку один-до-багатьох між різні пристроями. Остання функція означає, що датчикам і іншим вбудованим пристроїв, які переміщаються і взаємодіють зі шлюзом пристроїв, не потрібно знати, хто відправляє їм дані. Вони просто відправляють маршрут даних, і ті, хто підписується на дані, отримають його. Це забезпечує масштабовану середовище для малої затримки, низькою службової навантаження і двобічної зв'язку. Внутрішній шлюз пристроїв вбудований в повністю керовану і високодоступних середу, керовану спільнотою Amazon, щоб спростити розробку додатків і забезпечити уніфіковані заходи безпеки для всіх користувачів. Безпечний зв'язок між пристроями IoT і додатками гарантується, оскільки повідомлення MQTT передаються по протоколу TLS (Transport Layer Security), наступнику SSL (Secure Socket Layer). Крім того, шлюз пристроїв підтримує протокол HTTP 1.1.

З іншого боку, шлюз пристроїв об'єднується з іншим компонентом, званим механізмом правил. Механізм правил обробляє вхідні опубліковані повідомлення, а потім перетворює і доставляє їх на інші підписані пристрою або

хмарні сервіси AWS, а також сервіси не-AWS через AWS Lambda для подальшої обробки або аналізу. Це дає можливість створювати додатки IoT, які організовують, збирають, обробляють, аналізують і обробляють дані, що генеруються і публікуються підключеними пристроями по всьому світу, без необхідності звертати увагу на мережеві протоколи низького рівня або керувати будь-якої інфраструктурою. Щоб забезпечити зручність використання, розробники можуть створювати правила і додавати їх в обробник правил шляхом написання SQL-подібних операторів або використання служби консолі управління AWS. Правило складається з двох основних сегментів: оператора SQL і списку дій. Оператор SQL ідентифікує теми публікації / підписки, до яких застосовується правило, і умови, при яких має виконуватися правило. Список дій визначає набір дій, які повинні бути виконані при виконанні оператора SQL. У визначеннях правил використовується схема на основі JSON.

Правила поводяться по-різному в залежності від вмісту кожного вхідного повідомлення. Крім цього, ядро правил пропонує десятки вбудованих допоміжних функцій і обчислень для агрегації, перетворення, об'єднання та обробки даних і створення дуже складних правил. Розробники можуть створювати свої власні функції і визначати інші, використовуючи AWS Lambda. Механізм правил може отримувати дані з декількох джерел, з різних пристроїв і навіть із хмари AWS. Він інтегрує і направляє цю інформацію в інші пристрої IoT і хмарні сервіси AWS, такі як Amazon Kinesis, Amazon S3, Amazon DynamoDB і т. Д.

Модуль реєстру відповідає за призначення унікального ідентифікатора кожному підключеному пристрою незалежно від типу пристрою, постачальника або способу підключення. Крім того, він зберігає метадані (наприклад, ім'я пристрою, Id, атрибути і т. Д.) Підключених пристроїв, щоб мати можливість відстежувати їх. Якщо пристрій більше не активно і не відображається в мережі протягом 7 років, термін дії метаданих закінчиться і вони будуть видалені з реєстру. Консоль управління IoT AWS або інтерфейс командного рядка AWS можна використовувати для взаємодії з реєстром і його настройки вручну.

AWS IoT створює екземпляр кожного підключеного пристрою, створюючи віртуальний образ з ім'ям Device Shadow. Ця тінь постійна і зберігається в хмарі,

щоб бути доступною і доступною постійно. Він являє останній стан пристрою, коли воно було підключено до мережі, і встановлює майбутній стан фізичного пристрою, коли воно знову з'явиться в мережі. Це означає, що хмарні сервіси та інші пристрої можуть інтегрувати, зв'язуватися і зчитувати поточний стан певного пристрою через його тінь, навіть якщо пристрій знаходиться в автономному режимі. Вони також можуть оновлювати стан пристрою. Оновлення застосовуються, коли пристрій підключається до мережі. Читання останнього повідомленого стану і установка бажаного майбутнього стану виконується шляхом взаємодії з тінями пристроїв через REST API або за допомогою процесора правил. Ця функціональність допомагає легко контролювати пристрої і виконувати над ними дії без необхідності знати про низький рівень підключення. Це означає, що тінь прискорює розробку додатків, надаючи уніфікований і доступний інтерфейс для пристроїв, навіть коли вони використовують різні протоколи зв'язку і безпеки IoT або навіть коли вони обмежені переривчастим з'єднанням, обмеженою шириною смуги, обмеженої обчислювальної здатністю або обмеженими сила. З точки зору програмування, Тінь пристрої - це документ JSON, який використовується для зберігання та вилучення поточного стану певного пристрою.

При бажанні додатки можуть обмінюватися даними безпосередньо з підключеними фізичними пристроями, використовуючи тільки шлюз пристроїв і механізм правил. Це означає ігнорування реєстру і тіні пристрою. Тим не менш, це не рекомендується, оскільки користувач повинен зосередитися на підтримці базових протоколів зв'язку та вирішенні проблем синхронізації між підключеними пристроями і хмарою.

AWS IoT надає Device SDK, який дозволяє пристрою легко синхронізувати свій стан зі своєю тінню і приймати необхідні майбутні стану. Зокрема, AWS IoT Device SDK - це набір бібліотек, які допомагають підключати апаратні пристрої, виконувати аутентифікацію в хмарі, встановлювати мобільні додатки і легко обмінюватися повідомленнями. Він підтримує різні мови програмування, включаючи C і JavaScript.

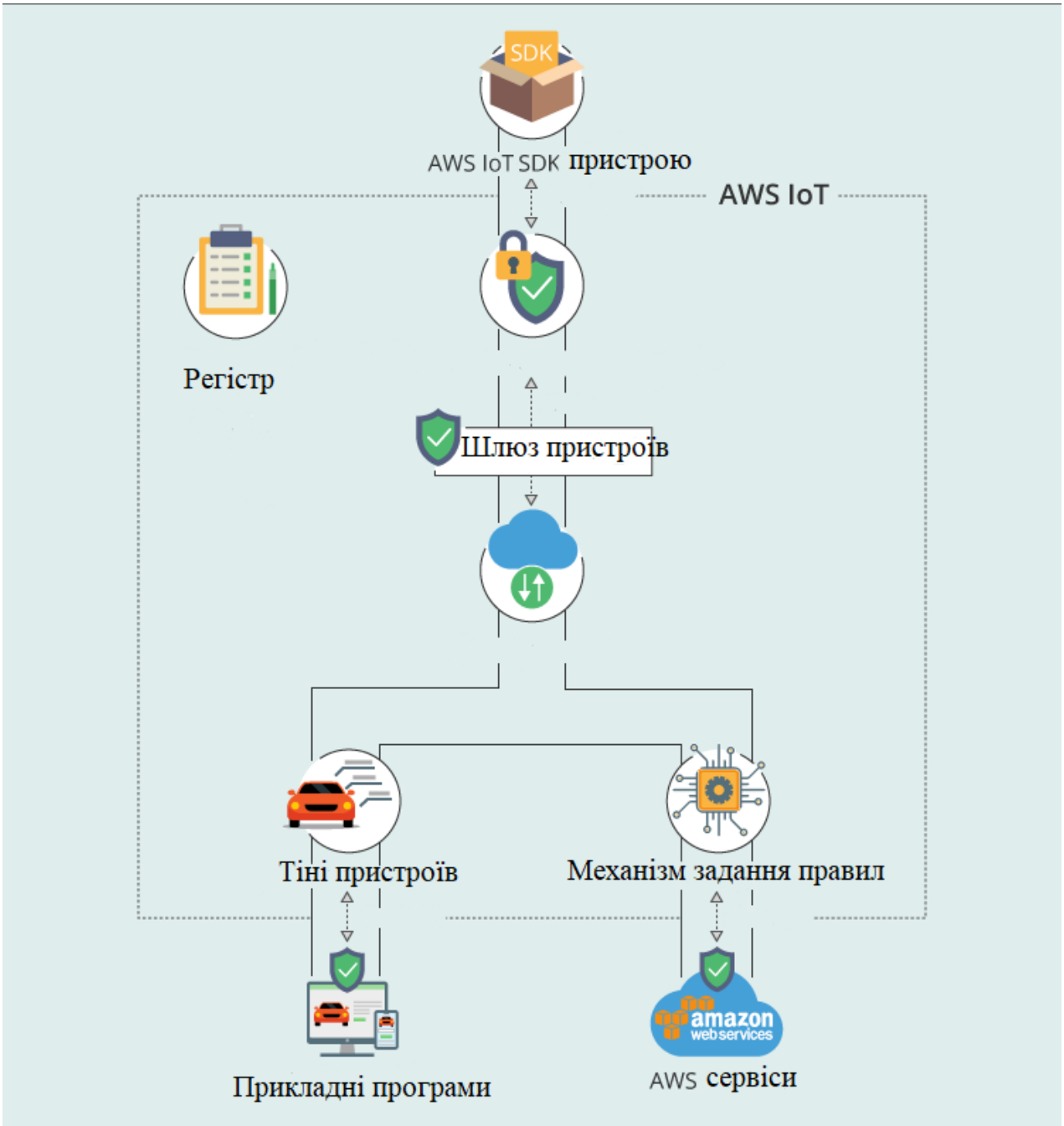

Рисунок 2.4.1 – Архітектура AWS IoT Platform[8]

2.4.2 Безпека як послуга з хмар (SeCaaS). Комплексний аналіз для використання в умовах IoT

У моделі «безпека як послуга» основна увага приділяється безпеці, що надається у вигляді хмарних послуг, тобто безпека забезпечується через хмару замість локальних рішень безпеки. Модель «безпека як послуга» може також поліпшити функціональність існуючих локальних реалізацій, рацюючи як гібридне рішення. У статті пропонується комплексний аналіз моделі надання

безпеки як послуги з різних точок зору. У документі розглядаються різні питання, пов'язані з безпекою, що надається як хмарний сервіс. У цьому документі розглядаються різні питання, які можуть виникнути щодо безпеки як послуги, а саме. визначення «безпека як послуга», її потреба, різні доступні в даний час послуги безпеки, різні постачальники послуг, різні користувачі і ймовірні користувачі цієї послуги, різні критерії оцінки рішень «безпека як послуга» і способи реалізації таких послуг. рішення.[4]

Модель безпеки як послуги фокусується на безпеці що надається у вигляді хмарних сервісів; тобто безпека забезпечується через хмару замість локальних рішень безпеки. Модель «безпека як послуга» може також поліпшити функціональність існуючих локальних реалізацій, працюючи як гібридне рішення. У документі пропонується комплексний аналіз безпеки як

Модель розміщеної безпеки стає переконливою альтернативою локальним рішенням безпеки на платформі. Як повідомляється в, загальна вартість володіння значно знижується в порівнянні з рішеннями для забезпечення безпеки в приміщеннях, оскільки в цих типах послуг відсутні попередні капітальні витрати. Подібні рішення знизять вимоги до локальної пропускної здатності і сховища, а також пов'язані з ними витрати. Багато рішень безпеки для віддалених і мобільних користувачів тепер стануть доступні. З поширенням більшої кількості кінцевих точок (мобільних і інших) і переміщенням шкідливого ПО також в хмару, загальна частота виявлення буде краще і ефективніше в порівнянні з результатами одного локального серверу, що працює на кінцевій точці. Модель «безпека як послуга» при впровадженні в організаціях покладе край дорогим захисного спорядження і складного розгортання програмного забезпечення для забезпечення безпеки. Хостингова модель безпеки може використовуватися гібридним чином для поліпшення функціональності існуючих локальних реалізацій. Таким чином, надаючи організаціям більше гнучкості у виборі бажаних варіантів безпеки для різних віддалених приміщень або штаб-квартири. Наприклад, організація може розгорнути локальну систему безпеки для роботи в штаб-квартирі, одночасно використовуючи розміщену систему безпеки для віддалених офісів. Сервіси і рішення будуть більш ефективними і простими для адміністрування, оскільки сервіси переходять в хмару як сервіс. Хостингові

рішення для забезпечення безпеки забезпечують кращу безпеку, оскільки вони використовують кілька систем сканування на наявність шкідливих програм в порівнянні зі скануванням на окремі шкідливі програми в локальних рішеннях. Вони оновлюють нові сигнатури і інші засоби захисту швидше в порівнянні з прямими оновленнями на різних кінцевих точках клієнтів в локальних рішеннях. Хмарні сервіси легко доступні та легко масштабовані і швидко розгортаються для негайного рішення безпекових питань.

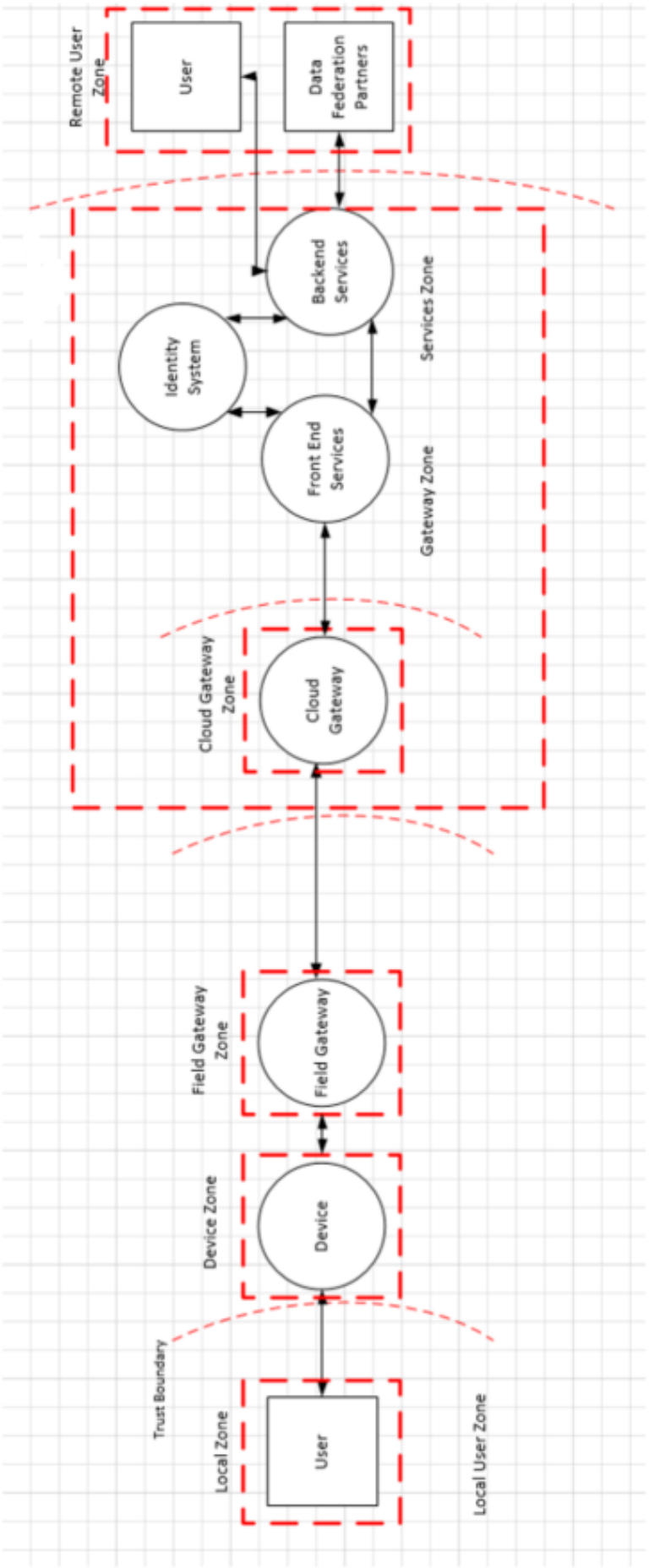

Рисунок 2.4.1 – Схематичне зображення ідеї SeCaaS

Альянс Cloud Security Alliance (CSA) визначив різні категорії пропозицій «безпека як послуга» у 2015 році:

• Управління ідентифікацією і доступом: включає управління доступом до корпоративних ресурсів шляхом перевірки особистості об'єкта і надання йому правильного рівня доступу на основі його авторизованого рівня.

• Запобігання втрати даних. Запобігання втрати даних - це захист і захист даних на різних етапах хмарної середовища. дані в спокої, в русі і при використанні як в хмарі, так і локально.

• Веб-безпека: веб-безпека - це захист в режимі реального часу, пропонована через хмару, шляхом перенаправлення веб-трафіку постачальнику хмари і наступного пересилання чистого трафіку в організацію клієнта.

• Захист електронної пошти. Захист електронної пошти забезпечує контроль над вхідною та вихідною електронною поштою, тим самим захищаючи організацію від фішингу, шкідливих вкладень і застосовуючи корпоративні політики відповідно до вимог організації клієнта.

• Оцінки безпеки: це аудити, що проводяться третьою стороною для хмарних сервісів, або оцінки локальних систем за допомогою хмарних рішень, заснованих на деяких галузевих стандартах.

• Управління вторгненням: Управління вторгненням - це процес виявлення / запобігання вторгнень з використанням сигнатурного або аномального підходу для реагування на незвичайні події.

• Управління інформацією про безпечність та подіями (SIEM): SIEM аналізує і зіставляє журнали і інформацію про події, пов'язані з проблемами безпеки, для надання звітів в режимі реального часу і сповіщень про інциденти / події безпеки, які можуть вимагати уваги.

• Шифрування. Це процес надання криптографічних алгоритмів з закритим і відкритим ключем для захисту даних в спокої, в русі і при використанні як в хмарі, так і в приміщеннях.

• Безперервність роботи та відновлення після збою. Це процеси і заходи, що забезпечують відмовостійкість у разі будь-яких збоїв і перерв в обслуговуванні.

• Мережева безпека: мережева безпека складається з положень безпеки, які розподіляють доступ, розподіляють, контролюють і захищають базові служби

мережевих ресурсів.

Основними областями яким виділяють найбільше уваги є захист кінцевих точок, захист Інтернету, оцінка і управління уразливими, управління ідентифікацією, захист від шкідливих програм і спаму, що надається різними постачальниками. Інші з'являються розміщені служби безпеки включають в себе керовані між мережеві екрани, IDS і IPS, безпеку обміну миттєвими повідомленнями, аутентифікацію, архівування електронної пошти та управління уразливими на основі хмари і т. Д. Безпека в якості послуги, ймовірно, буде продовжувати рости в майбутньому не тільки з точки зору можливості безпеки, але також з точки зору різних варіантів, пропонованих послуг і їх інтенсивності.

2.5 Висновки до розділу 2

В розділі була приведена історія розвитку використання IoT та найбільш можливе майбутнє. На основі чого було розглянуто, проаналізовано та розписано можливі методи захисту мереж. Відповідно до основних особливостей була проведена робота по можливості використання їх.

У випадку з взаємопов’язаними речами повинні бути розробленні рішення на основі шифрування трафіку, який передають між собою пристрої. Така парадигма є найбільш можливою для майбутнього, де взаємопов’язані речі будуть домінувати. Але на разі використання не є можливим, через те що абсолютна більшість пристроїв не мають необхідної можливостей для роботи з ключами шифрування більшими ніж 64 біта з цих же причин не можливе використання технології blockchain.

На даний момент є популярним ідея туманних сервісів. По суті є проміжною ланкою та чимось схожа на хмарні сервіси. Основною проблемою цього рішення є його незавершеність, для розуміння це аналог 5G. Всім зрозуміло що він значно краще, але конкретних реалізацій не було знайдено також не вдалося навіть купити одне з таких рішень для його вивчення. Також в наукових працях та статтях було помічено тренд збільшення затримок, що значно погіршує QoS. Тому туманною є не тільки ідея але й її реалізація.

В кінці було показано ідею захисту в хмарах. Так наприклад розглянуто впровадження SeCaaS. Яка має декілька переваг та є найпростішою в реалізації. Більш за все приваблює фактичного зняття відповідальності та перекладу її на вендора послуг. Але на шляху використання є декілька характеристик що ускладнюють роботу. Неоднорідність мереж є одою з таких дуже важко структурувати та великий потік не оброблених даних. Також є проблемами втрати зв’язку, втрати даних та їх зберігання.

РОЗДІЛ 3. ЕКОСИСТЕМА ДЛЯ КОНВЕРГЕНЦІЇ МЕРЕЖ IOT ТА БЕЗПЕКОВИХ РІШЕНЬ SECAAS

3.1 Архітектурна реалізація системи

В даному розділі буде обґрунтовано логічний сенс та кожен компонент запропонованої архітектури. Структурно вона зображена на рисунку 3.1 Також на рисунку було вказано допоміжні підсистеми та сторонні додатки (бази даних, брокер сервери), які було використано в різних модулях екосистеми. Такий вибір буду прояснено в наступних розділах роботи.

Дані надходять в систему від множини пристроїв IoT. Формат даних та основі їх характеристики різняться. Ці данні надсилаються до первинних серверів також пристрої передають автентифікаційну інформацію та мета дані про себе.

Первинні сервери виконують перевірку автентифікаційної інформації. У разі неправильної інформації (тобто інформацію було надіслано з невідомого джерела), дані не будуть опрацьовані. В залежності які дані надіслав пристрій, на первинних серверах дані групуються: з’ясовується тип даних та розподіляється до визначеної категорії в чергу з даними (модуль «Apache Kafka»). За сумісністю первинні сервери проводять процедуру валідації даних тобто перевірки їх цілісності.

Архітектура екосистеми

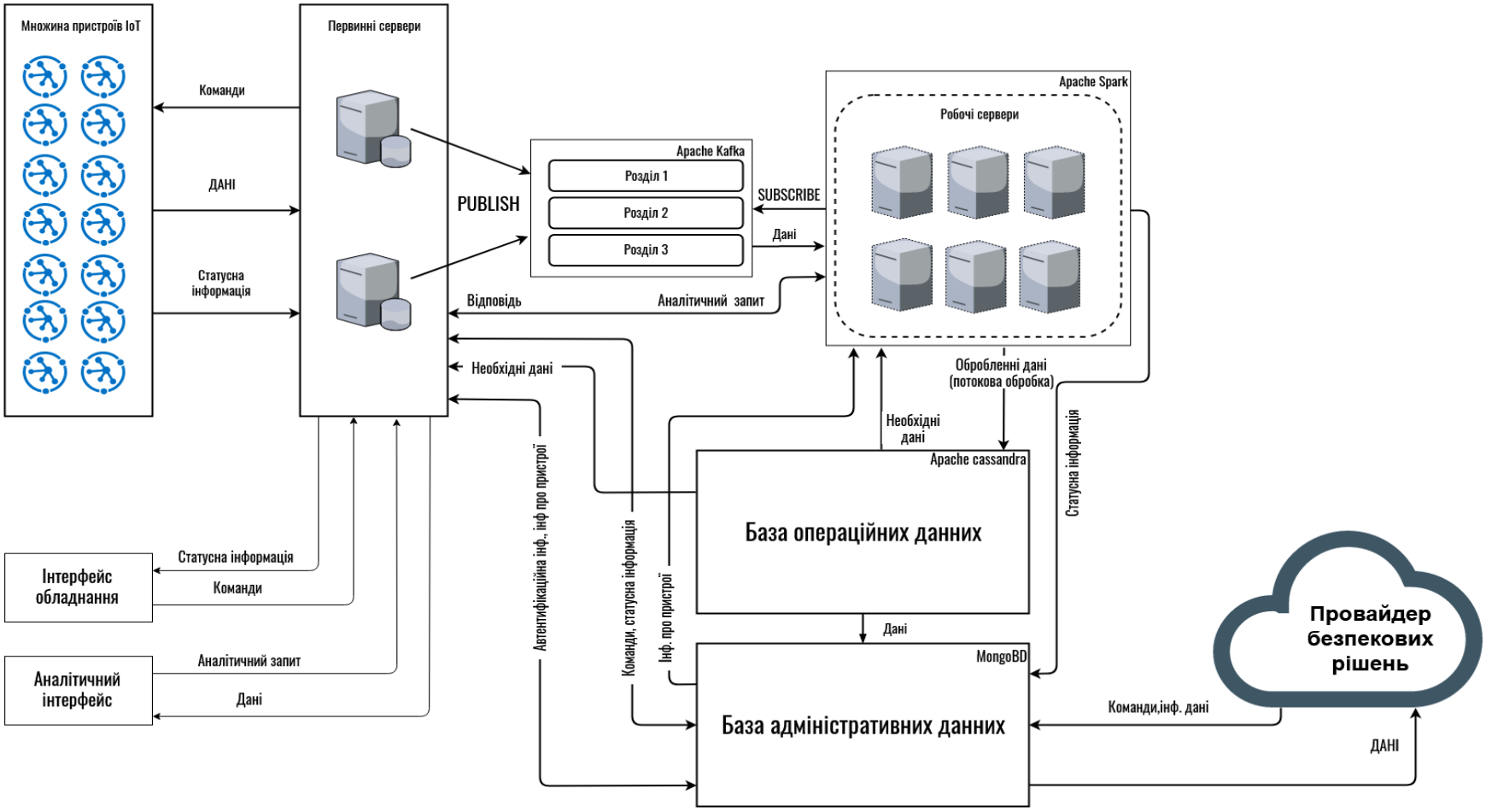

Рисунок 3.1 – Запропонована архітектура екосистема для IoT мереж

Черга даних виконує патерн проектування publish-subscribe.

Первинні сервери розмішують (publish) дані у Apache Kafka.

На боці subscribe розташовано систему обробки даних – Apache Spark. Вона виконує структурування даних, які потім записуються у базі операційних даних. Робота з даними на цій ітерації є потоковою або мікропакетною (micro-batch). В разі мікропакетного варіанту можливо емулювати потокову обробку, тобто стає можливою застосовуваня інструментів і логіки пакетної обробки. Ще одним аргументом за є те, що мікропакетна обробка сама по собі відповідає логіці агрегування та віконної (windowing) обробки, в разі необхідності зібрати деяку кількість даних за хвилину, обробити їх певним чином та записати до бази даних.

База даних, в якій зберігаються операційні дані, забезпечує безперебійний доступ до даних, в разі якщо до них потрібен доступ. По критерію продуктивності, це сховище має бути оптимізоване під запис, через те що сценаріїв зчитування буде значно менше. На випадок, якщо база операційних даних буде не в змозі записати дані (у разі різкого росту навантаження), можливо використати чергу повідомлень, як буфер. Потрібно відзначити, що вимоги до цієї бази, не такі як більш звичні транзакційні БД.

На наступному рахунку буде описано порядок проходження інформації від множини пристроїв Iot до баз даних на постійне зберігання. Схематично це зображено на рисунку 3.2

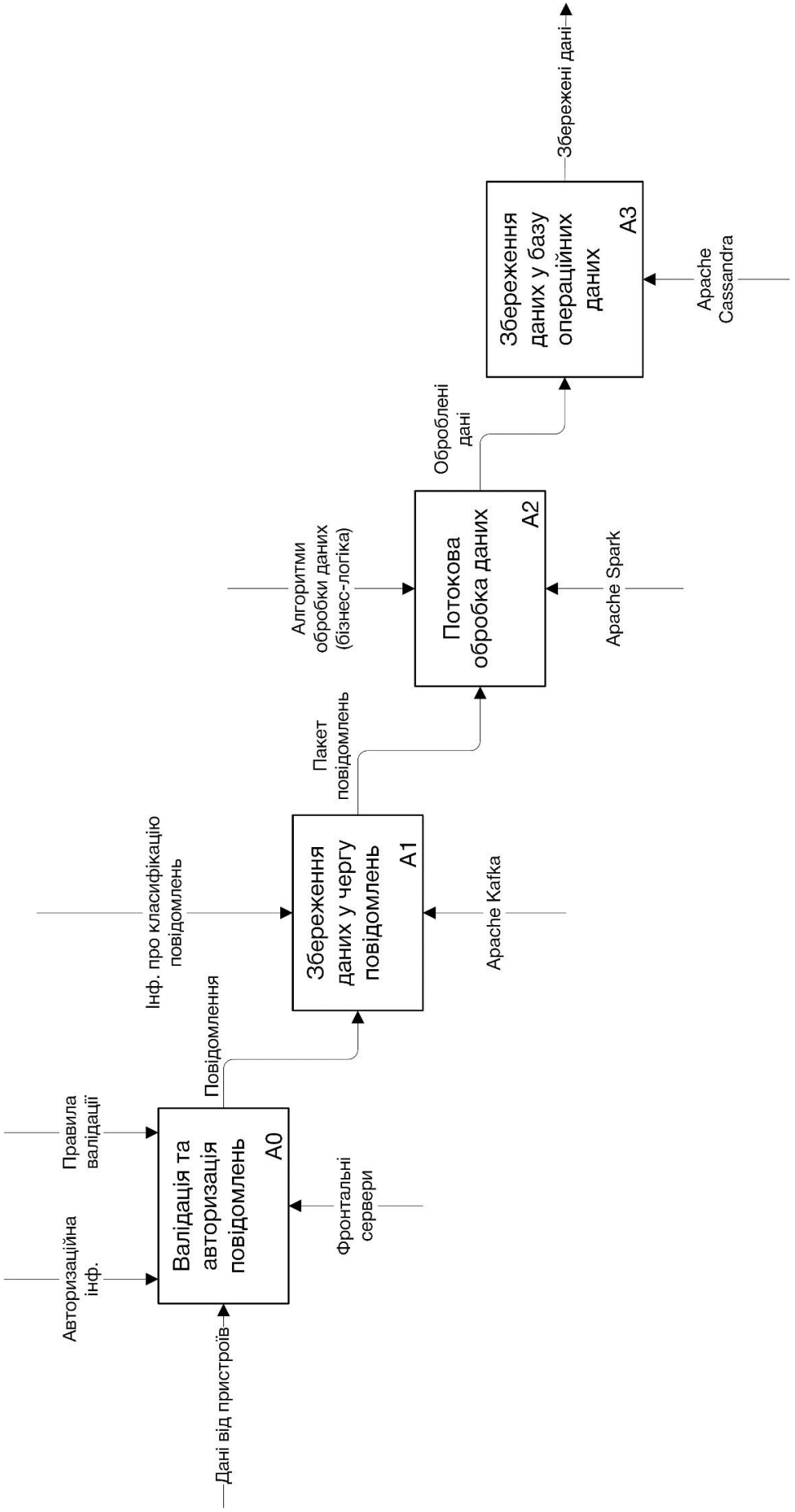

Рисунок 3.2 – Діаграма потоків даних і процесів при отриманні від пристроїв

Проговорим передачу даних від пристрою до адміністратора. Такими даними може бути статусна інформація, наприклад, заряд батареї пристрою або деякі повідомлення про збій.

Статусні дані надходять від пристроїв до первинних серверів, на цьому кроці повторюються валідація та автентифікація адресанта запиту. Потім інформація надсилається до бази адміністративних даних.

В такому разі частіше всього необхідні ресурсоємні гарантії постійного зберігання та узгодженості інформації. Тому після того, як первинний сервер запише дані до бази даних, він буде чекати підтвердження їх отримання та зберігання перед тим, як відправити потрібне підтвердження пристрою, який повинен виконати логіку повторного запиту в разі, якщо повідомлення є важливим.

В разі збереження цієї інформації адміністратор відправляє запит до первинних серверів, які звіряють його права доступу до цієї інформації та роблять запит до даних відповідно предикату запиту, як в разі статусних повідомлень пристрою що вказують на його ID).

Потоки даних для цих ітерацій циклу представлено на рисунку. 3.3

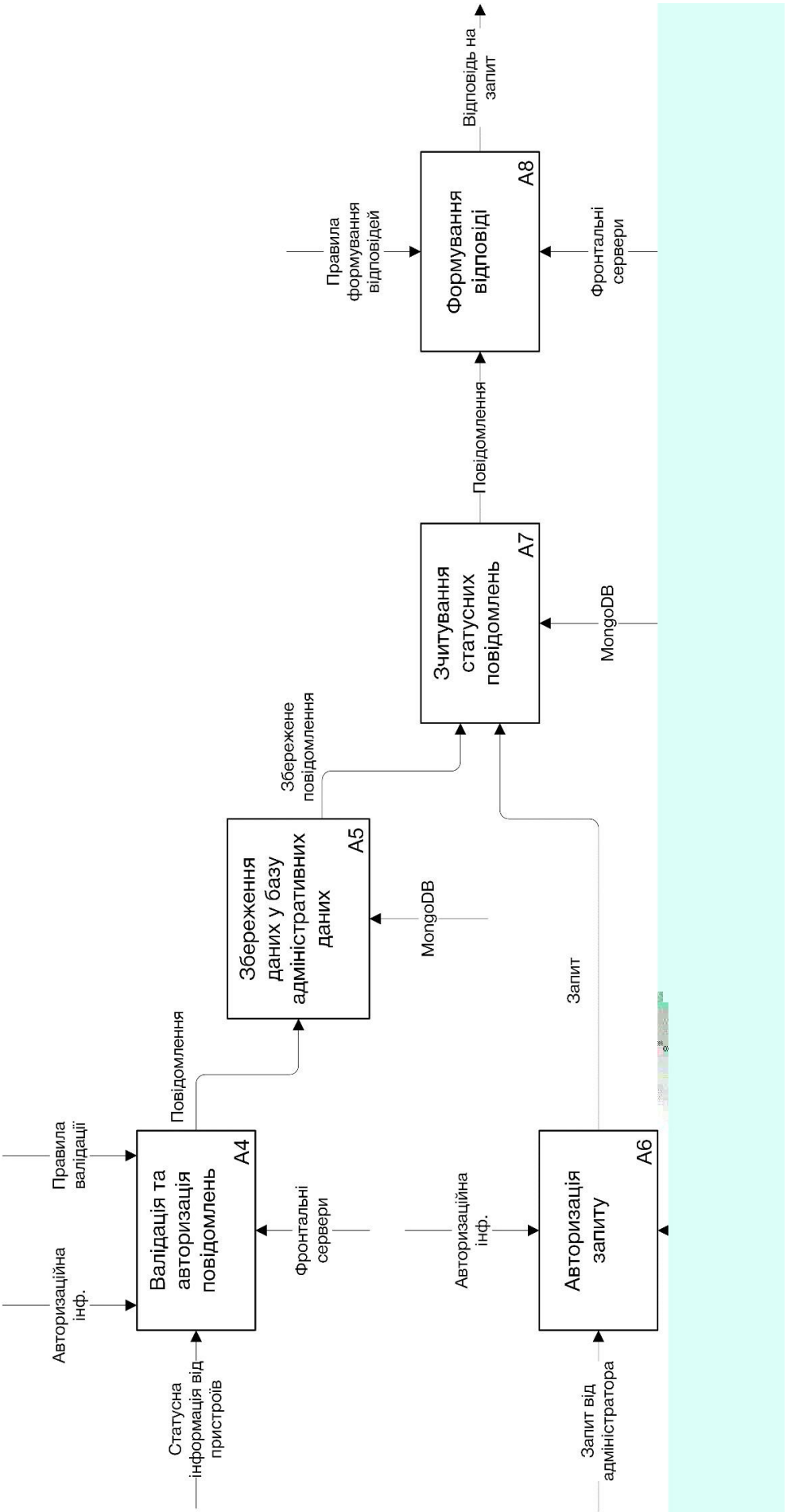

Рисунок 3.3 – Діаграма потоків даних і процесів при передачі статусної інформації від пристрою до адміністратора

Наступним статусна інформація надходить з механізмів моніторингу. Адміністратори у форматі правил створюють чіткі умови, які мають виконуватися при очікуваній роботі пристрою, та детермінують порядок по якому і відбувається перевірка.

Тобто, адміністратор може назначити правило, що деякий пристрій повинен посилати не менше встановленої кількості повідомлень за встановлений проміжок часу. Це означає, що якщо ця вимога не виконується, тобто приходить в дію певний тригер, платформа посилає статусне повідомлення, для того щоб повідомити адміністратора про несправність.

Подібні тригери можуть спрацьовувати в різних модулях системи. первинні сервери можуть слідкувати за статусом з'єднання з пристроями. Модуль обробки даних може час від часу (за розкладом) відсилати запит до сховища операційних даних для того, щоб відфільтрувати можливо некоректну інформацію, спричинену некоректну роботою пристрою.

Опишемо передачу команд від адміністратора до пристрою. Керуючий сигнал від адміністратора приходить до первинного сервера, який гарантовано зберігає її в базі адміністративних даних.

Пристрої із встановленою періодичністю опитують первинні сервери на випадок появи нових керуючих сигналів. Потоки даних показано на рис. 3.4.

Така архітектура необхідна для того, що б гарантовано доправляти інформацію до пристроїв, на випадок якщо вони будуть тимчасово недоступні (відключені від живлення або не втратили зв'язок з сервером).

Теоретично, покращити параметр затримки передачі даних і оптимізувати використання ресурсів можна було б, відсилаючи дані без посередніх операцій (якщо таке можливо). Наприклад, якщо первинний сервер має пряєм з'єднання з пристроєм (по WebSocket), то при отриманні команди від адміністратора можна відправляти її одразу. Але при цьому буде зменшено гарантії доставки. Якщо команду занесено до бази даних, то пристрій може отримати її, виконати, і тільки після цього відправити підтвердження первинному серверу, який позначить команду як прийняту чи зітре її. У разі чіткого відправлення команди IoT пристрою, він відзначить її отримання, але в разі збою, пристрій не матиме змоги ще раз отримати цю команду до первинного серверу, бо її буде втрачено. В разі

не підтвердження отримання команди до її виконання, то адміністратор повинен буде значний час очікувати завершення цього запиту, що негативно вплинуло б на його сприйняття інтерфейсу в порівняні з асинхронною схемою, коли адміністратор отримує підтвердження після того, як ця команда записана в базі даних. Тепер буде прояснено дії, які відбуваються при отриманні аналітичних запитів. Первинні сервери автентифікують запит і перевіряють тип запиту. Запит може бути виконаний з наявними даними або вимагати додаткових операцій. Наявні дані – це дані які пройшли первинну обробку та зберігаються в базі операційних даних. При цьому виду запиту досить перевіряти відповідні дані та відправляти їх.

Якщо ж прийшов аналітичний запит, то необхідно створити план його реалізації, зчитати дані з першою або з двох баз даних, а після обробити їх. Якщо необхідна тільки додаткове опрацювання наявних даних, то на вхід системи обробки затребується лише дані з бази операційних даних. База адміністративних даних може використовуватися, в разі коли запит потребує зчитування певної інформації про пристрої. Скажімо, аналітику будуть потрібні агрегована інформація від пристроїв на певній ділянці території. В такому випадку необхідно спочатку знайти в базі адміністративних даних ID пристроїв на цій ділянці, а потім зчитати операційні дані від належних сенсорів за їх ID.

Потоки даних у сценарії виконання аналітичних запитів виглядають так, як показано на діаграмі (рис. 3.4).

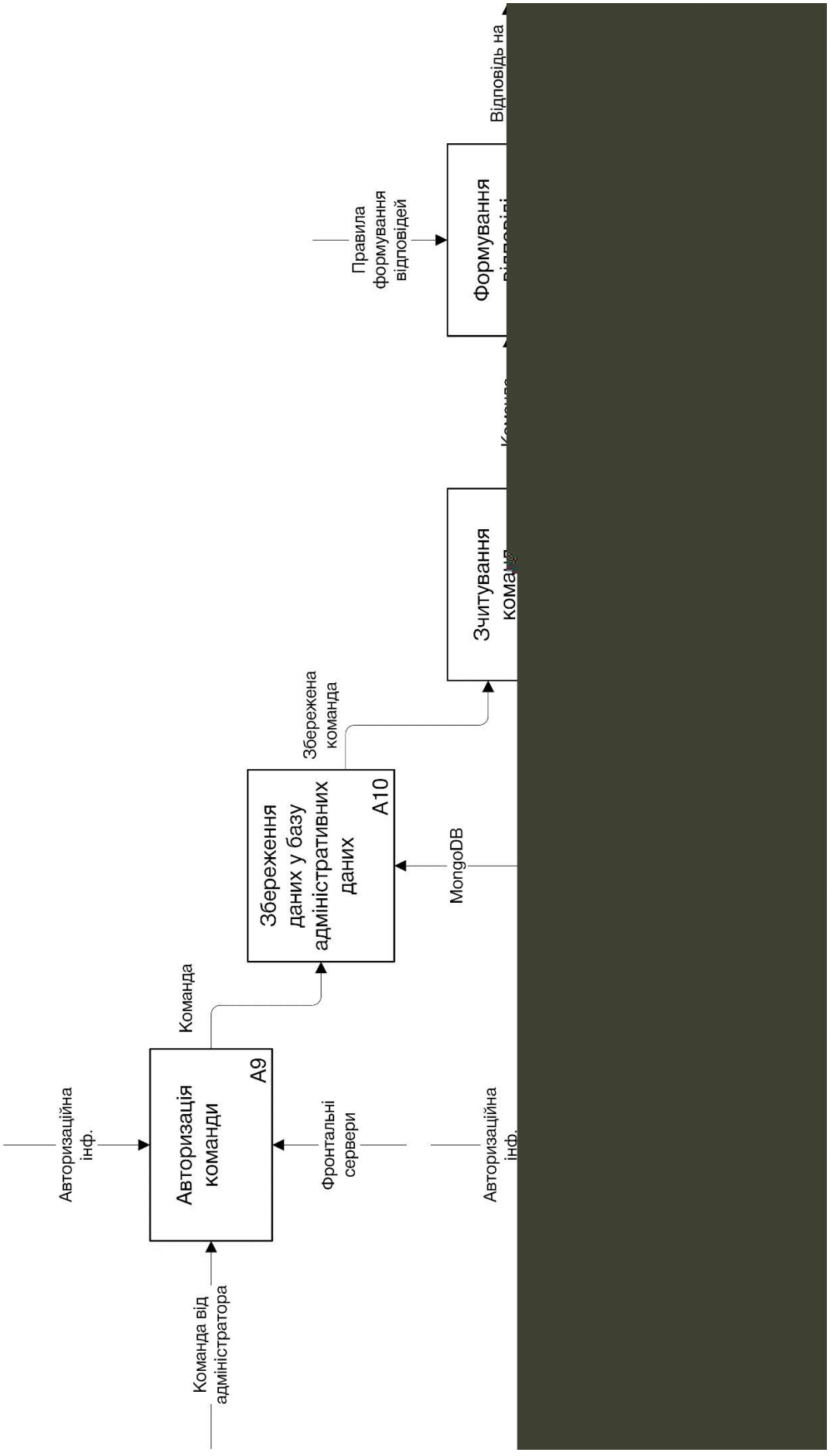

Рисунок 3.4 – Діаграма потоків даних і процесів при передачі команд від адміністратора до пристрою

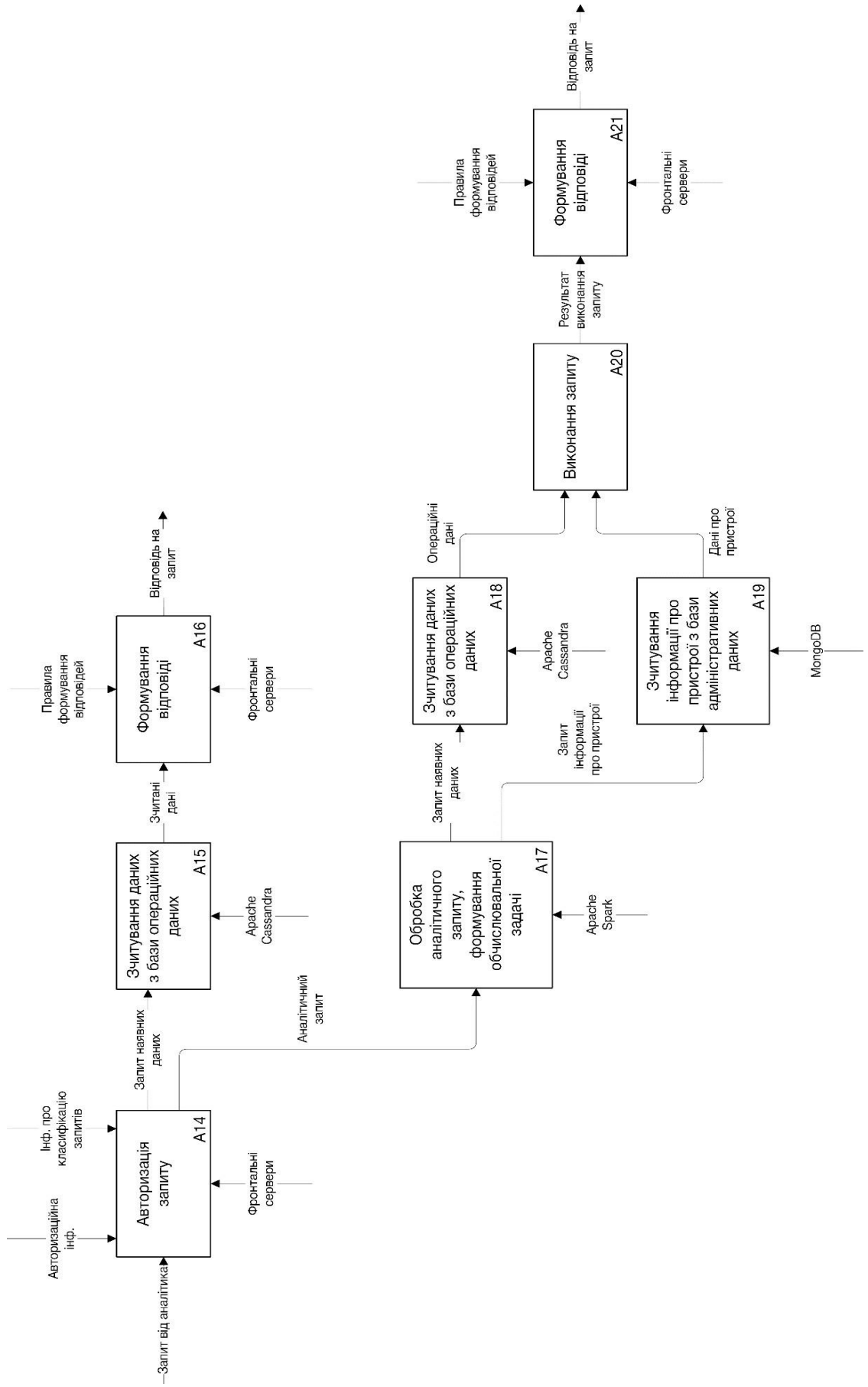

Рисунок 3.5 – Діаграма потоків даних і процесів при обробці запитів від аналітика

3.2 Програмні модулі використані для екосистеми

3.2.1 Apache Kafka – черга повідомлень для прийому операційних даних

Apache Kafka – це брокер повідомлень із відкритим вихідним кодом,розроблений Apache Software Foundation і написаний на Scala. Kafka – розподілений, партиціонований (розділений), реплікований журнал записів (commit log). Він пропонує функціональність системи повідомлень, але має унікальну архітектуру [20].

Спочатку коротко опишемо термінологію:

- Стрічка повідомлень ділиться на категорії, що називаються темами (topic).
- Процеси, які публікують (publish) повідомлення у Kafka називаються постачальниками (producer).
- Процеси, які підписуються (subscribe) на теми та обробляють стрічку опублікованих повідомлень, називаються споживачами або приймачами (consumer).
- Kafka працює як кластер, що охоплює один або більше серверів.

3.2.2 Apache Spark – система обробки даних

Apache Spark – це система обробки великих даних із відкритим вихідним кодом, побудована для забезпечення високої швидкості роботи, простоти використання та складної аналітики [21]. Розроблена у 2009 році у AMPLab (UC Berkeley), публічний реліз як Apache проекту відбувся у 2010 році.

Spark має значні переваги порівняно з іншими Big Data і MapReduce технологіями, такими як Hadoop і Storm.

Spark дає всебічний та уніфікований каркас для роботи з різними за своєю сутністю і джерелом наборами даних.

3.2.3 Apache Cassandra – база даних для зберігання операційних

Apache Cassandra - це вільно поширюване розподілене сховище з широкими стовпцями з відкритим вихідним кодом, система управління базами даних NoSQL, розроблена для обробки великих обсягів даних на багатьох звичайних серверах, забезпечуючи високу доступність без єдиної точки відмови. Cassandra пропонує надійну підтримку кластерів, що охоплюють кілька центрів обробки даних, з асинхронної репликацією без майстра, що дозволяє виконувати операції з низькою затримкою для всіх клієнтів.

3.2.4 MongoDB – база даних для зберігання адміністративних даних

MongoDB – крос-платформна документо-орієнтована база даних. Класифікована як NoSQL, MongoDB замість традиційних табличних структур реляційних баз використовує схожі на JSON документи з динамічною структурою (формат називається BSON), що робить інтеграцію даних швидшою. Розроблена MongoDB Inc. і опублікована як безкоштовне ПО із відкритим вихідним кодом під комбінацією ліцензій GNU Affero General Public License і Apache License.

Оглянемо основні риси MongoDB:

Ad-hoc запити. Підтримуються запити полів, діапазонів, пошук з регулярними виразами. Запити можуть зчитувати окремі поля документів та включати задані користувачем JavaScript функції, які виконуються безпосередньо сервером.

Індексування. Проіндексувати можливо будь-яке поле в документі, включно з масивами та вкладеними документами (індекси в MongoDB концептуально схожі на індекси в реляційних СУБД). Доступні первинні та вторинні індекси.

Реплікація. Доступність підтримується за рахунок наборів реплік (replica sets). Набір реплік складається із двох або більше копій даних. Кожен такий набір може виконувати роль первинної або вторинної репліки. Первинна за замовчанням виконує всі записи та читання, тобто забезпечується сильна

узгодженість. Вторинні підтримують копію даних первинної репліки. У разі збою первинної репліки, набір реплік автоматично обирає, яка з вторинних реплік стане первинною. Вторинні репліки можуть (опціонально) виконувати операції читання, але дані будуть консистентні в остаточному рахунку.

Балансування навантаження. MongoDB горизонтально масштабується за допомогою шардінгу. Користувач обирає ключ, який визначає, як розподіляються дані в колекції. Вони діляться на діапазони та розподіляються по кількох серверах. Ключ шардінгу також може хешуватися для рівномірного розподілу.

Агрегація. Дозволяє виконувати запити, аналогічні SQL GROUP BY. Оператори агрегування можуть поєднуватися в ланцюжки, як Unix pipes. Також є оператор lookup, який об’єднує документи.

Швидкі операції видалення. MongoDB зберігає дані у двобічно зв’язаному списку, тому для видалення потрібно лише змінити два вказівника. Але, автоматична компактифікація відсутня, тому місце на диску не вивільнюється автоматично, а реалізується окремою утилітою.

Обмежені колекції. Підтримують колекції фіксованого розміру, що називаються обмежені колекції (capped collections). Вони підтримують порядок вставки, та при досягненні вказаного розміру поводяться як циклічна черга (circular queue). Можуть бути корисними для статусних повідомлень і команд, наприклад, аби не зберігати більше вказаного числа записів. Використання таких колекцій може бути набагато ефективнішим за регулярне видалення записів.

Відсутність підтримки багатодокументних транзакцій. ACID транзакції підтримуються лише на рівні документа. Але, як буде показано пізніше, структура документів, що буде використовуватися, не вимагає багатодокументних транзакцій.

Динамічна структура. Оскільки значення в документах зберігаються разом із назвами полів, можна динамічно змінювати модель даних.

Повнотекстовий пошук. Підтримуються досить широкі можливості повнотекстового пошуку, вимагає наявності текстового індексу.

3.3 Протоколи і формати обміну даними між компонентами системи

Опишемо протоколи передачі даних між компонентами системи.

Зв’язок між девайсами і платформою відбувається за допомогою зашифрованого протоколу HTTP/2, який реалізує криптографічний протокол TLS 1.2. Шифрування трафіку потрібне, аби уникнути перехоплення даних та інших атак. Протокол HTTP/2, стандартизований [17] у 2015 році, дуже добре підходить для IoT, бо:

- Розроблений як оптимізація HTTP/1.1.
- Реалізує стискання HTTP-заголовків (HPACK) за допомогою коду Хаффмана. Цей метод має високу ефективність, але потребує мало пам’яті, що дуже важливо для малопотужних пристроїв.
- За допомогою таких технік як pipelining та мультиплексування заохочує використання одного TCP-з’єднання, що дозволяє уникати повільного тристороннього рукостискання (three-way handshake), яке, до того ж, потребує додаткових ресурсів. У зашифрованому протоколі перевикористання з’єднання дозволяє уникнути ще більш довгого TLS діалогу (negotiation).
- Передбачає операцію PING на рівні протоколу, що дозволяє використовувати її для перевірки працездатності з’єднання з пристроєм.
- Підтримує високорівневу сумісність із HTTP/1.1: методи, коди стану, URI, поля заголовків.

Багато платформ Інтернету речей використовують MQTT, який є набагато ефективнішим за HTTP/1.1. Проте із появою HTTP/2 за продуктивністю HTTP/2 і MQTT майже зрівнялися.

Головною перевагою використання HTTP є його поширеність і загальноприйнятість у Web, що означає наявність функціональних і стабільних серверних і клієнтських бібліотек.

HTTP/2 реалізує техніку server push, яка дозволяє серверу ініціювати відправку даних клієнту замість традиційної моделі «запит-відповідь». Проте

більш природнім і ефективним для повнодуплексного зв’язку є застосування протоколу WebSocket [18]. Пристрої, якщо вони мають можливість установити з’єднання з сервером, підтримують відкритий канал, по якому можуть отримувати команди адміністраторів від сервера за протоколом WebSocket. На практиці використовують його захищену варіацію WebSocket – WebSocket Secure (WSS).

У якості основного формату даних, у якому фронтальні сервери отримують інформацію, використовується JSON. Він, як і HTTP, є дуже поширеним; є читабельним для людини (тобто не є бінарним); є простим для формування та розділу.

Загалом, на первинних серверах реалізується REST API, який використовується пристроями, адміністраторами та аналітиками.

Як уже було зазначено, зв’язок між пристроями та точкою входу в хмарну інфраструктуру платформи відбувається з використанням зашифрованого з’єднання. Дешифрування (SSL termination) відбувається на рівні розподільника навантаження, і первинні сервери отримують уже розшифровані дані.

Наочно формати даних і протоколи їх передачі між різними компонентами системи показано на рис. 2.6.

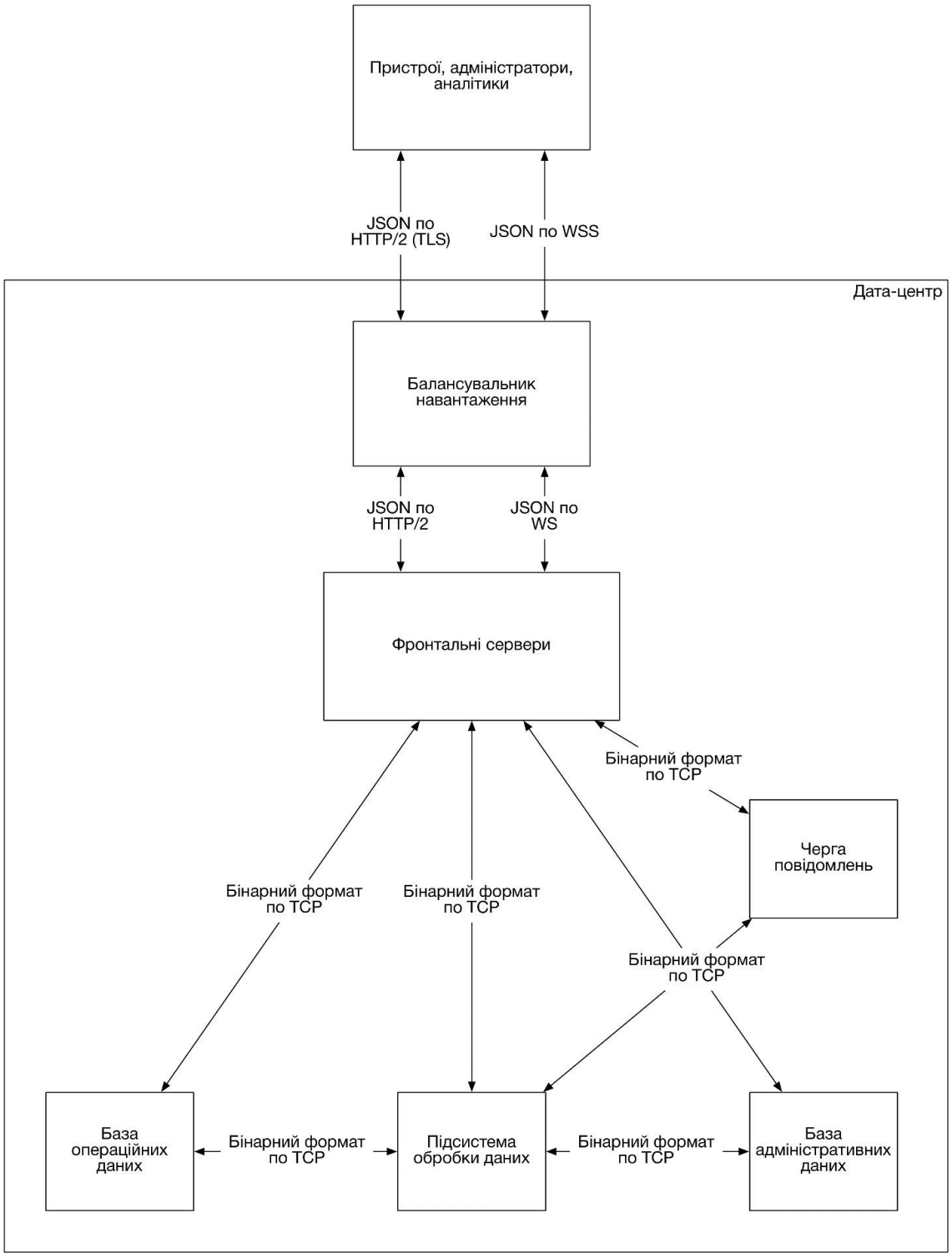

Рисунок 3.3.1 – Запропоновані формати даних і протоколи їх передачі між компонентами системи

3.4 Підсистема обробки даних. Організація, вимоги та обґрунтування вибору стороннього рішення

Описана в підрозділі «Архітектура платформи» схема обробки даних реалізує так звану лямбда-архітектуру (lambda architecture). Розглянемо цей концепт детальніше.

Лямбда-архітектура – архітектура обробки даних, створена для роботи з великими обсягами даних, використовуючи одночасно переваги підходів пакетної та потокової обробки даних. Цей підхід до архітектури намагається збалансувати затримку, пропускну спроможність і відмостійкість, застосовуючи пакетну обробку для надання всебічних та точних розрізів (view) пакетів даних, одночасно використовуючи потокову обробку в режимі реального часу для створення розрізів операційних даних. Ці два розрізи даних можуть бути об’єднані перед виводом. Поширення лямбда-архітектури обумовлене збільшенням обсягів даних, потребами аналітики реального часу та намаганням зменшити затримки MapReduce.

Лямбда-архітектура працює з моделлю даних, яка має незмінюване (до якого дані лише дописуються, тобто append-only) джерело. Ця модель призначена для запису та обробки подій із мітками часу, які дописуються до існуючих даних замість перезапису. Стан системи визначається природнім упорядкуванням даних по часу.

Лямбда-архітектуру зображено на діаграмі (рис. 2.7)

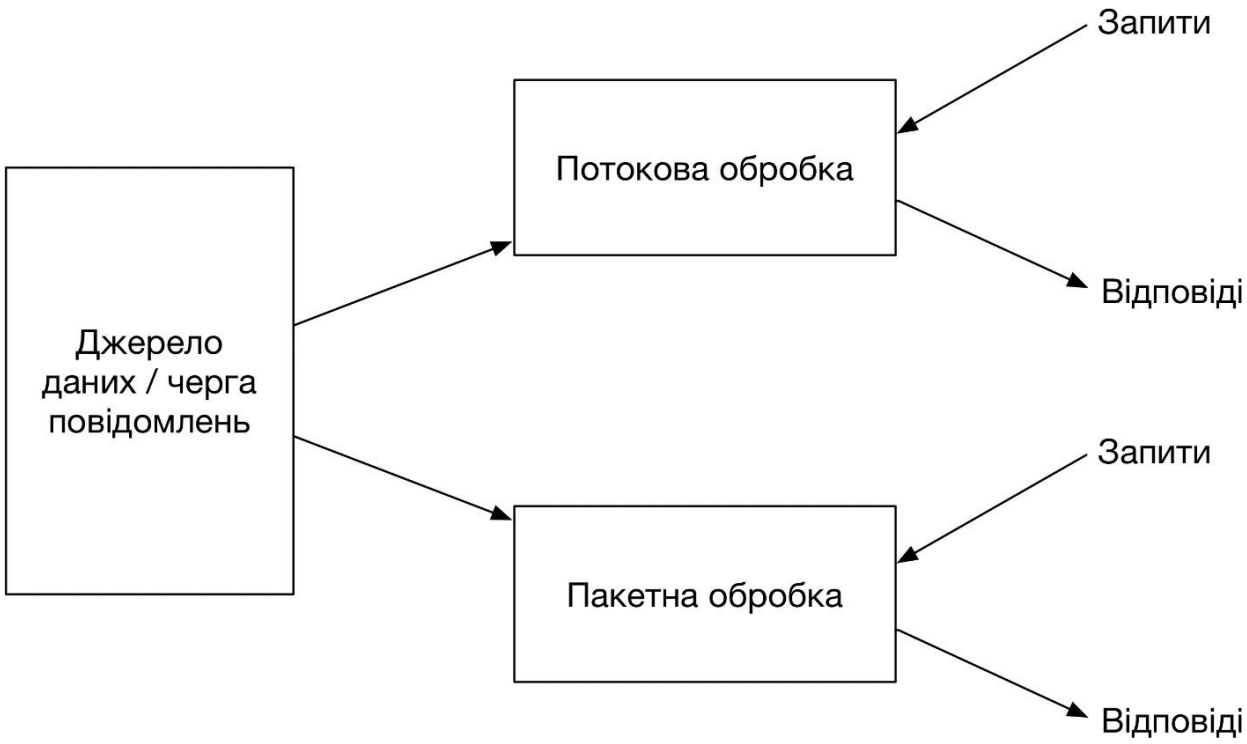

Рисунок 3.4.1 – Лямбда-архітектура

Опишемо також підхід MapReduce, який частково застосовується для обробки даних платформою. Цей підхід призначений для обробки задач, які паралелізуються. Він складається із таких кроків:

- Map**:** кожен worker-сервер застосовує функцію map до даних і записує результат у тимчасове сховище. Головний сервер гарантує, що оброблена буде лише одна копія вхідних даних.
- Shuffle**:** worker-сервери розподіляють дані відповідно до вихідних ключів (виходів функції map) так, щоб усі дані, які відповідають одному ключу, розташовувались на одному worker-сервері.
- Reduce; worker-сервери обробляють кожну групу даних, відповідно до ключа, паралельно.

3.5 Висновки до розділу 3

Рішення у вигляді екосистеми що представлене в цьому розділі конвергує SeCaaS, та усуває основні перешкоди на шляху його використання.

Якщо раніше розглядували тільки два рівня хмара та самі пристрої то запропоноване рішення надає можливості по розділу на деякі підрівнів в залежності від специфікації ресурсів та набору політик визначаючих використання різних пристроїв на цих під рівнях. Також є можливість легко масштабувати та спрощувати взаємодію між IoT пристроями та SeCaaS вендорами.

Серед багатьох можливостей потрібно відмітити збільшення незалежності від зовнішніх факторів. Так є операційне сховище та база адміністративних даних, що уберігає від ризиків втрати зв’язку або розторгання контракту з вендором послуг. Так надає буфер часу для закриття цих питань.

Велику роль відіграє брокер даних та інструменти для впорядкування інформації, що дозволяє адаптувати реальність інтернету речей до строго структурованого вигляду та впровадити основне рішення SeCaaS. Великим зрушенням є те що дві сторони не повинні бути активними постійно що зменшує оперативні витрати.

ВИСНОВКИ

В даній дипломній роботі було проведену детальний опис технології IoT. Глибоку аналітику проблематики з описом основних проблем в безпеці досліджуваних мереж. На основі отриманих результатів та знань про галузь безпеки в IoT, було запропоноване комплексне рішення для застосування хмарних методів безпеки на основі сервісів SeCaaS.

На початку захищати IoT планувалося різними методами по ходу виконання роботи були знайдені проблеми що змушували відмовитися від тих чи інших рішень. Так наприклад, спочатку планувалося впроваджувати рішення на основі технології блок чейну та шифрування, але значна кількість пристроїв які випускають не має потужностей для обчислення криптографічних ключів довжиною більш ніж 32 біта. Таким чином відпадає можливість рішення питання цим шляхом. В майбутньому IoT повинен перетворитися пов’язані між собою речі, які не потребують керуючих серверів(безсерверна архітектура).

Після була розглянута концепція Fog що обіцяла вирішення всіх проблем в безпеці. При всій перспективності технології та великої кількості наукових статей практичної реалізації знайдено не було, навіть змоги купити якесь готове рішення або заказати під вихідні дані не дало ніяких результатів. До того ж було знайдено роботи що вказують на значне збільшення затримок, що призводить до погіршенню QoS. Через ці проблеми було вирішено відмовитися від застосування даної технології.

Під кінець було вирішено використовувати рішення SeCaaS, перевагами даної технології є простота в реалізації та перекладання відповідальності за безпеку на вендора. До проблем можемо віднести високу степінь залежності від провайдера послуг, складність при агрегації даних з пристроїв, необхідне постійне підключення до інтернету та вразливість перед загрозою нульового дня.

Основним перешкодами на шляху використання хмарного рішення

SeCaaS були не сумісність з неоднорідними мережами IoT та хаотичним і складно аналізуємого трафіку що генерують мережі. Таким чином для вирішення даних питань було створено систему для конвергенції досліджуваних мереж із SeCaaS послугами.

Рішенням даних проблем є створення екосистеми для кластеризації, аналізу та зберігання даних після чого вони будуть переслані до провайдеру послуг.

Таким чином не потрібно постійного підключення до мережі інтернет. У разі зміни політики вендором завжди є можливість змінити його без втрати безпекового компоненту за рахунок бази адміністративних даних.

Великою перевагою запропонованого вирішення э простота при масштабуванні мереж, що потребує не значного налаштування після встановлення нового обладнання. Особливістю цієї системи є можливість динамічний зв’язок багато до багатьох та асинхронний зв'язок.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Secure integration of IoT and Cloud Computing/ Christos Stergiou, Kostas E. Psannis, Byung-Gyu Kim, Brij Gupta 2016p.
2. InternetofThings:AsurveyonthesecurityofIoTframeworks/ MahmoudAmmara,GiovanniRussellob,BrunoCrispo 2017p.
3. The Web of Things: interconnecting devices with high usability and performance/ Simon Duquennoy, Jean-Jacques Vandewalle. 2019p.
4. Towards Security as a Service (SecaaS): On the modeling of Security Services for Cloud Computing/ Angelo Furfaro ; Alfredo Garro ; Andrea Tundis 2014p.
5. EiF: Toward an Elastic IoT Fog Framework for AI Services/ JongGwan An, Wenbin Li, Franck Le Gall, Ernoe Kovac, Jaeho Kim, Tarik Taleb, and JaeSeung Song. 2019p.
6. From the Internet of Computers to the Internet of Things/ Friedemann Mattern and Christian Floerkemeier 2009p.
7. Scalable and Configurable End-to-End Collection and Analysis of IoT Security Data/ Aikaterini Roukounaki, Sofoklis Efremidis, John Soldatos, Juergen Neises, Thomas Walloschke, Nikos Kefalakis 2016p.
8. Інформація про платформу AWS IoT. – Режим доступу: https://aws.amazon.com/iot/how-it-works/.
9. Офіційна документація Apache Kafka. – Режим доступу: http://kafka.apache.org/documentation.html. –
10. Офіційна документація Apache Spark. – Режим доступу: http://spark.apache.org/docs/latest/. –
11. Офіційна документація Apache Cassandra. – Режим доступу: https://wiki.apache.org/cassandra/.
12. Офіційна документація MySQL. – Режим доступу: http://dev.mysql.com/doc/refman/5.7/en/.
13. Офіційна документація Amazon Web Services. – Режим

доступу: https://aws.amazon.com/documentation/.

14. Офіційна документація Docker. – Режим доступу: https://www.docker.com/.

ДОДАТОК А
Лістинг програми

MessageQueueSender.java

```
package com.iot.mq;

import java.util.concurrent.CompletionStage;

public interface MessageQueueSender {
        CompletionStage<Void> send(String topic, String key, String data);
```

DoubleStatsCounter.scala

```
package com.iot.util

import scala.math.sqrt

case class DoubleStats(count: Long, avg: Double, stdDev: Double)

case class DoubleStatsCounter(var n: Long = 0, var sum: Double = 0.0, var
sumOfSquares: Double = 0.0) {
  def add(number: Double):
    DoubleStatsCounter = { n += 1
    sum += number
    sumOfSquares +=
    number * number this
  }
  def merge(anotherCounter: DoubleStatsCounter):
    DoubleStatsCounter = { n += anotherCounter.n
    sum += anotherCounter.sum
    sumOfSquares += anotherCounter.sumOfSquares this
  }
  def avg = sum / n
  def stdDev = sqrt(sumOfSquares / n - avg * avg)
  def stats = DoubleStats(n, avg, stdDev)
}
object DoubleStatsCounter {
  def apply(number: Double): DoubleStatsCounter = new
  DoubleStatsCounter().add(number)
}
```

```
package com.iot.mq;

import org.apache.kafka.clients.producer.KafkaProducer; import
org.apache.kafka.clients.producer.Producer; import
org.apache.kafka.clients.producer.ProducerRecord;

import org.apache.kafka.common.serialization.StringSerializer;
import java.util.Properties;

import java.util.concurrent.CompletableFuture;
public class KafkaSender implements MessageQueueSender { private final
      Producer<String, String> producer;
      public KafkaSender() {
    Properties properties = new Properties();
    properties.put("bootstrap.servers", "localhost:9092");
    properties.put("acks", "1");
    properties.put("key.serializer", StringSerializer.class.getName());
    properties.put("value.serializer", StringSerializer.class.getName()); producer =
    new KafkaProducer<>(properties);
    }
public CompletableFuture<Void> send(String topic, String key, String data) {
        ProducerRecord<String, String> record = new ProducerRecord<>(topic,
key, data);
                CompletableFuture<Void> future = new
                CompletableFuture<>(); producer.send(record,
                (metadata, exception) -> {
                        if (exception != null) {
                                future.completeExceptionally(exception);
                        } else {
                                future.complete(null);
                        }
                });
```

```
package com.iot.auth;

import com.mongodb.async.client.MongoClient; import
com.mongodb.async.client.MongoClients; import
com.mongodb.async.client.MongoCollection; import
com.mongodb.async.client.MongoDatabase; import
org.bson.Document;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.CompletionStage;
import static com.mongodb.client.model.Filters.eq;

public class MongoAuthenticationService implements AuthenticationService {
        public static final String DB_NAME = "iot";
        public static final String COLLECTION_NAME = "device_info";
        public static final String DEVICE_ID =
        "device_id"; public static final String
        PASSWORD = "password";

        private final MongoCollection<Document>

        deviceInfoCollection; public

        MongoAuthenticationService() {
             MongoClient mongoClient = MongoClients.create();
            MongoDatabase db = mongoClient.getDatabase(DB_NAME);
            deviceInfoCollection = db.getCollection(COLLECTION_NAME);
        }
        @Override
      public CompletionStage<Boolean> authenticate(String login, String
            password) { CompletableFuture<Boolean> future = new
            CompletableFuture<>(); deviceInfoCollection.find(eq(DEVICE_ID,
            login)).first((result,
exception) ->
{                           if (exception != null) {
                                    future.completeExceptionally(exceptio
                                    n); return;
```

```
package com.iot.token;

import com.auth0.jwt.JWTSigner;
import com.auth0.jwt.JWTVerifier;

import java.util.Map;
import java.util.regex.Pattern;

public class JwtTokenService {
	private static final String SECRET = "feRtfVqoyDJWVVoP4X2m";

	private static final Pattern AUTH_HEADER_PATTERN =
Pattern.compile("^Bearer$", Pattern.CASE_INSENSITIVE);

	private final JWTVerifier verifier = new
	JWTVerifier(SECRET); private final JWTSigner signer =
	new JWTSigner(SECRET);

	public String createToken(Map<String, Object>
		claims) { return signer.sign(claims);
	}

	public Map<String, Object> verifyToken(String authorizationHeader)
throws TokenParseException,
		TokenVerificationException {
		String token = extractToken(authorizationHeader);
		try {return verifier.verify(token);
		} catch (Exception e) {
			throw new TokenVerificationException(e);
		}
	}

	private static String extractToken(String authorizationHeader)
throws TokenParseException {
		if (authorizationHeader == null)
	throw new TokenParseException("Authorization header value is null");

		String[] parts = authorizationHeader.split(" ");
		if (parts.length != 2) {
			throw new TokenParseException("Incorrect format: '" +
authorizationHeader
+ 		"'. Format is Authorization: Bearer

[token]");
		}
```

HttpUtils.java

```
package com.iot.http;
import io.undertow.server.HttpServerExchange;
import java.util.Deque;
import java.util.Map;
import static io.undertow.util.StatusCodes.BAD_REQUEST;
import static io.undertow.util.StatusCodes.INTERNAL_SERVER_ERROR;
public class HttpUtils {
public static Void sendRequestError(HttpServerExchange exchange) {
        return sendError(exchange, BAD_REQUEST);
         }
         public static Void sendServerError(HttpServerExchange
          exchange) { return sendError(exchange,
          INTERNAL_SERVER_ERROR);
         }
         public static Void sendError(HttpServerExchange exchange, int
        code) { return sendError(exchange, "", code);
         }

         public static Void sendError(HttpServerExchange exchange, String body, int
code)
{          exchange.setStatusCode(code);
           exchange.getResponseSender().send(body); return null;
```

AuthenticationService.java

```
package com.iot.auth;
import
java.util.concurrent.CompletionStage;
public interface
AuthenticationService {
        CompletionStage<Boolean> authenticate(String login, String password);
```

```
package com.iot.http;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.iot.auth.AuthenticationService;
import
com.iot.token.JwtTokenService;
import
io.undertow.server.HttpHandler;
import
io.undertow.server.HttpServerExchange;
import io.undertow.util.Headers;
import io.undertow.util.Methods;
import
io.undertow.util.StatusCodes;

import java.util.Deque;
import
java.util.HashMap;
import java.util.Map;
import java.util.concurrent.CompletionStage;

import static
com.iot.http.HttpUtils.extractQueryParameter; import
static com.iot.http.HttpUtils.sendError;
import static com.iot.http.HttpUtils.sendServerError;
import static
io.undertow.util.StatusCodes.UNAUTHORIZED;

public class AuthenticationHttpHandler implements
        HttpHandler { public static final String DEVICE_ID
        = "deviceId"; public static final String LOGIN =
        "login";
        public static final String PASSWORD = "password";

        private final AuthenticationService
        authenticationService; private final JwtTokenService
        tokenService;
        private final ObjectMapper objectMapper;

        public AuthenticationHttpHandler(AuthenticationService
authenticationService, JwtTokenService tokenService,
                                         ObjectMapper objectMapper) {
                this.authenticationService = authenticationService;
                this.tokenService = tokenService;
                this.objectMapper = objectMapper;
        }

        @Override
        public void handleRequest(HttpServerExchange exchange) throws
```

```
package com.iot.http;

import com.fasterxml.jackson.databind.JsonNode; import
com.fasterxml.jackson.databind.ObjectMapper; import
com.iot.mq.MessageQueueSender;
import com.iot.token.JwtTokenService; import
com.iot.token.TokenParseException;
import com.iot.token.TokenVerificationException; import
io.undertow.server.HttpHandler;
import io.undertow.server.HttpServerExchange; import
io.undertow.util.HeaderValues;
import io.undertow.util.Headers; import
io.undertow.util.Methods; import
io.undertow.util.StatusCodes;

import java.io.IOException; import java.util.Map;

import static com.iot.http.AuthenticationHttpHandler.DEVICE_ID;
import static com.iot.http.HttpUtils.*;
import static com.iot.mq.KafkaTopics.DATA_TOPIC;
import static io.undertow.util.StatusCodes.UNAUTHORIZED;

public class DataHttpHandler implements HttpHandler {
        public static final String DATA = "data";

        private final MessageQueueSender
        sender; private final JwtTokenService
        tokenService; private final ObjectMapper
        objectMapper;

        public DataHttpHandler(MessageQueueSender sender, JwtTokenService
tokenService, ObjectMapper objectMapper) {
                this.sender = sender;
                this.tokenService = tokenService;
                this.objectMapper = objectMapper;
        }
```

```
package com.iot.http;

import
com.fasterxml.jackson.databind.ObjectMapper;
import com.iot.auth.AuthenticationService;
import
com.iot.auth.MongoAuthenticationService;
import com.iot.mq.KafkaSender;
import com.iot.token.JwtTokenService;
import com.iot.weather.http.ReportHttpHandler;
import io.undertow.Undertow;
import static io.undertow.Handlers.routing; public class HttpServer {
        public static void main(final String[] args) {
                JwtTokenService tokenService = new
                JwtTokenService(); ObjectMapper
                objectMapper = new ObjectMapper();
                AuthenticationService authenticationService =
                new
MongoAuthenticationService();
                DataHttpHandler dataHttpHandler = new
DataHttpHandler(new KafkaSender(), tokenService,
objectMapper);
                AuthenticationHttpHandler authenticationHttpHandler =
        new AuthenticationHttpHandler(authenticationService,
        tokenService, objectMapper);
        ReportHttpHandler reportHttpHandler = new ReportHttpHandler();

        Undertow server = Undertow.builder()
        .addHttpListener(8080, "localhost")
        .setHandler(routing()
        .get("/api/report/device/{id}",
        reportHttpHandler)
        .post("/api/data", dataHttpHandler)

                        .get("/api/auth/login/{login}/password/{password}",
                                        authenticationHttpHandler))
                                .build();

        server.start();
```

WeatherDomain.scala

```
package com.iot.weather

import java.util.Date
import com.iot.util.{DoubleStats,
DoubleStatsCounter} object WeatherDomain
{
  // db
  val (keyspace, table) = ("iot", "weather")
  // columns
  val (deviceId, timestamp, temperatureStats, pressureStats) =
("device_id", "timestamp", "temperature_stats", "pressure_stats")
  val (count, avg, stdDev) = ("count", "avg", "std_dev")
  case class AggregateWeatherDataRecord(temperatureStats:
  DoubleStatsCounter,
    pressureStats: DoubleStatsCounter) { def merge(anotherRecord:
    AggregateWeatherDataRecord) = {
    temperatureStats.merge(anotherRecord.temperatureStats)
    pressureStats.merge(anotherRecord.pressureStats)
      this
    }
  }
  case class WeatherDatabaseRecord(deviceId: String, timestamp: Date,
 temperatureStats: DoubleStats, pressureStats:
DoubleStats)
  case class WeatherDataRecord(temperature: Double, pressure: Double)
}
```

ReportHttpHandler.scala

```
package com.iot.weather.http

import com.datastax.spark.connector._
import com.iot.http.HttpUtils.{extractQueryParameter,
sendServerError} import com.iot.weather.WeatherDomain._
import io.undertow.server.{HttpHandler, HttpServerExchange}
import io.undertow.util.HttpString
import org.apache.spark.{SparkConf,
SparkContext} import
org.json4s.NoTypeHints
import
org.json4s.jackson.Serialization
import
org.json4s.jackson.Serialization._

import
scala.concurrent.ExecutionContext.Implicits.global
import scala.util.{Failure, Success}

class ReportHttpHandler extends HttpHandler { val conf = new
  SparkConf()
    .setMaster("local[*]")
    .setAppName(getClass.getSimpleName)
    .set("spark.cassandra.connection.host",

  "127.0.0.1") val sc = new SparkContext(conf)

  implicit val formats = Serialization.formats(NoTypeHints)

  val corsHeader = new HttpString("Access-Control-Allow-

  Origin") override def handleRequest(exchange:

  HttpServerExchange): Unit = {
    var weatherTable =
    sc.cassandraTable[WeatherDatabaseRecord](keyspace, table)
    Option(extractQueryParameter("id", exchange.getQueryParameters))
      .foreach(id => weatherTable = weatherTable.where(s"$deviceId
    = ?", id)) exchange.dispatch
    val rowsFuture =
    weatherTable.collectAsync()
    exchange.getResponseHeaders.add(corsH
    eader, "*") rowsFuture.onComplete {
      case Success(rows) =>
      exchange.getResponseSender.send(write(rows)) case
      Failure(e) => sendServerError(exchange)
    }
```

```
package com.iot.weather.stream

import java.util.Date

import com.datastax.spark.connector.cql.CassandraConnector
import com.datastax.spark.connector.streaming._
import com.iot.mq.KafkaTopics.DATA_TOPIC import
com.iot.util.DoubleStatsCounter import
com.iot.weather.WeatherDomain._ import
org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.kafka.KafkaUtils
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.{SparkConf, SparkContext}
import org.json4s._
import org.json4s.jackson.JsonMethods._

object WeatherStreamingProcessor extends App { implicit val formats =
  DefaultFormats
  val batchDuration = Seconds(1) val conf = new SparkConf()
    .setMaster("local[*]")
    .setAppName(getClass.getSimpleName)
    .set("spark.cassandra.connection.host", "127.0.0.1")

  val sc = new SparkContext(conf)
  val ssc = new StreamingContext(sc, batchDuration)

  type Key = String
  val (zkQuorum, groupId) = ("localhost:2181", "weather-consumer") val stats =
  "stats"
  // json fields
  val (temperature, pressure) = ("temperature", "pressure")

  CassandraConnector(conf).withSessionDo { session => session.execute(s"DROI
    KEYSPACE IF EXISTS $keyspace") session.execute(s"CREATE
    KEYSPACE IF NOT EXISTS $keyspace " +
    s"WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 1 }")
    session.execute(s"CREATE TYPE $keyspace.$stats ($count BIGINT, $avg
    DOUBLE, $stdDev
    DOUBLE)")
```

```
    session.execute(
      s"""CREATE TABLE IF NOT EXISTS $keyspace.$table
      ($deviceId TEXT, $timestamp TIMESTAMP,
      $temperatureStats FROZEN<stats>,
      $pressureStats FROZEN<stats>,
      PRIMARY KEY ($deviceId, $timestamp))""")
    session.execute(s"TRUNCATE $keyspace.$table")
  }
  val stream = KafkaUtils.createStream(ssc, zkQuorum, groupId,
Map(DATA_TOPIC -> 1), StorageLevel.MEMORY_ONLY)
  stream
    .map(parseKafkaRecord)
    .map(toAggregateRecord)
    .reduceByKeyAndWindow((r1: AggregateWeatherDataRecord, r2:
AggregateWeatherDataRecord) => r1.merge(r2),
      batchDuration, batchDuration)
    .map(toDatabaseRecord)
    .saveToCassandra(keyspace, table) ssc.start()
  def parseKafkaRecord(t: (String, String)): (Key,
    WeatherDataRecord) = { val json = parse(t._2)
    (t._1, WeatherDataRecord(
      (json \ temperature).extractOrElse(0.0), (json \
      pressure).extractOrElse(0.0)))
  }

  def toAggregateRecord(t: (Key, WeatherDataRecord)): (Key,
  AggregateWeatherDataRecord)
=
    (t._1, AggregateWeatherDataRecord(DoubleStatsCounter(t._2.temperature),
DoubleStatsCounter(t._2.pressure)))

  def toDatabaseRecord(t: (Key, AggregateWeatherDataRecord)) =
    WeatherDatabaseRecord(t._1, new Date(), t._2.temperatureStats.stats,
t._2.pressureStats.stats)
}
```