Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: «Розробка системи надання обчислювальних ресурсів на основі концепції туманних обчислень»

Виконав: студент VI курсу, групи СПм-61
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Костур П.М. (прізвище та ініціали)

Керівник (підпис) Михалик Д.М. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) ЛитвиненкоЯ.В. (прізвище та ініціали)

Тернопіль 2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 20___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Костуру Петру Михайловичу

1. Тема роботи Розробка системи надання обчислювальних ресурсів на основі концепції туманних обчислень

Керівник роботи Михалик Дмитро Михайлович, к.т.н., доц. каф. ПІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом по університету від « 4 » грудня 2023 року № 1141
2. Термін подання студентом роботи 26.12.2023
3. Вихідні дані до роботи наукові літературні джерела

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1 Аналіз предметної області дослідження. 2. Проектування рішення
3. Практична частина. 4. Охорона праці та безпека в надзвичайних ситуаціях

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема роботи. 2. Мета, задачі дослідження, об'єкт, предмет, наукова новизна
3. Концепція ТО як підтримка інфраструктури в хмарі. 4. Вимоги до системи.
5. Діаграма прецедентів системи. 6. Діаграма компонентів системи.
7. Діаграма послідовності додавання образу програми.
8. Діаграма послідовності запуску туманних додатків.
9. Програмне забезпечення та механізми, використані в розробці.
10. Сторінка репозиторіїв системи. 11. Сторінка зареєстрованих пристроїв туману.
12. Сторінка образів туманних додатків. 13. Сторінка API розробника.
14. Створення кластера IoT – пристроїв.
15. Основні результати проведеного дослідження.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Г.М., к. т. н., доц., зав. каф. КС		
Безпека в надзвичайних			
ситуаціях	Стручок В.С., ст. викл. каф. ОХ		

7. Дата видачі завдання _____ 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз предметної галузі дослідження	19.09–27.09.23	Виконано
2.	Обґрунтування актуальності дослідження	30.09–10.10.23	Виконано
3.	Огляд існуючих механізмів та засобів	11.10–28.10.23	Виконано
4.	Проведення дослідження механізмів та засобів опрацювання даних	29.10–15.11.23	Виконано
5.	Оформлення розділу «Аналіз предметної області дослідження»	16.11–28.11.23	Виконано
6.	Оформлення розділу «Дослідження та проектування рішення»	29.11–05.12.23	Виконано
7.	Оформлення розділу «Практична частина»	06.12–10.12.23	Виконано
8.	Оформлення розділу «Охорона праці та безпека в надзвичайних ситуаціях»	05.12–12.12.23	Виконано
9.	Нормоконтроль	13.12–15.12.23	Виконано
10.	Перевірка на плагіат	13.12–17.12.23	Виконано
11.	Попередній захист роботи	20.12.23	Виконано
12.	Захист кваліфікаційної роботи	27.12.23	

Студент _____ Костур П.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Михалик Д.М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 65 сторінок, 25 рисунків, 34 джерел та 1 додаток.

ВУЗОЛ ТУМАНУ, ЛАТЕНТНІСТЬ СИСТЕМИ, РОЗГОРТАННЯ СИСТЕМИ, ТУМАННА ПРОГРАМА, ТУМАННІ ОБЧИСЛЕННЯ, API, IoT, JSON

У кваліфікаційній роботі магістра проведено огляд та аналіз існуючих наукових поглядів та програмних засобів за темою дослідження, а також технологій підтримки туманних обчислень.

Наведено ключові варіанти застосування та архітектура розробленої туманної системи обчислень. Описано програмну реалізація складових частин системи із використанням мови Python та бази даних MongoDB. Описано процес її розгортання, відображено результати експериментального тестування ключових варіантів її використання. Також представлені основні результати поведеного дослідження та можливі напрями подальших досліджень.

The qualifying thesis contains: 65 pages, 25 figures, 34 sources and 1 append.

FOG NODE, SYSTEM LATENCY, SYSTEM DEPLOYMENT, FOG SOFTWARE, FOG COMPUTING, API, IoT, JSON

In the master's thesis, a review and analysis of existing scientific views and software tools on the topic of research, as well as technologies for supporting fog computing, was carried out.

The key application options and architecture of the developed fog computing system are presented. The software implementation of the component parts of the system using the Python language and the MongoDB database is described. The process of its deployment is described, the results of experimental testing of key variants of its use are displayed. The main results of the conducted research and possible directions of further research are also presented.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) – прикладний програмний інтерфейс

ID (Identifier) – ідентифікатор

IoT (Internet of Things) – Інтернет речей

JSON (JavaScript Object Notation) –текстовий формат обміну даними

HTTP(HyperText Transfer Protocol) — протокол передачі гіпертексту

P2P (Peer-to-Peer) – однорангова децентралізована мережа, основана на рівноправності учасників

Protocol Buffers – формат серіалізації даних від корпорації Google

REST (REpresentational State Transfer) – передача репрезентативного стану

RPC (Remote Procedure Call) – віддалений виклик процедур

БД – база даних

ВМ – віртуальна машина

ВТ – вузол туману

КП – кінцевий пристрій

Латентність – час доступу/затримки в системі між запитом до неї та моментом, коли доступ завершується або повертаються запитувані дані

ОС – операційна система

ОР – обчислювальні ресурси

ПЗ – програмне забезпечення

ПТ – пристрій туману

ТД – токен доступу

ТО – туманні обчислення

ТОС – туманна обчислювальна система

ХО – хмарні обчислення

ЦОД – центр обробки даних

Черга повідомлень – це форма асинхронної комунікації між сервісами, що застосовується в безсерверних та мікросервісних архітектурах

ФС – файлова система

ЗМІСТ

ВСТУП

Актуальність теми дослідження. В даний час дуже широко застосовуються ХО для різноманітних завдань: збереження файлів, спілкування, розважальний контент чи наукові здобутки. В такого підходу є величезні переваги, проте є і деякі обмеження. Найсерйозніше з них – латентність. Вона не дає змогу застосовувати ХО в тих галузях, де потрібен найбільш швидкий відгук системи при деяких подіях.

Фірми, котрі дають ОР на базі концепції ХО, просто не в можуть ґарантувати низьку латентність їх систем, тому що ОР можуть знаходитися дуже далеко від фінального юзера. Така проблема не здатна бути вирішеною звичайним збільшенням числа ЦОД компаній по всій земній кулі.

На сьогоднішній день при застосування ХО користувач має доступ до якихось ресурсів, які можуть знаходитися географічно далеко від нього, і досить часто юзер і не володіє інформацією, де розташоване те обладнання, на якому він працює. Це пов'язано з фізичним розміщенням центрів даних фірм, послуги якої замовник використовує. Є фірми, котрі володіють значною кількістю обладнання, розташованого по всій Землі, але навіть за такої умови, на одну країну припадає лише кілька ЦОД, що абсолютно не є вирішенням проблеми для системи значної її латентності.

Один із шляхів розв’язання цієї задачі – застосування концепції ТО, в основі котрої лежить той факт, що усі розрахунки проходять на обладнаннях туману (обчислювальному устаткуванні), які розміщені близько з точки зору географії до користувача. Іншими словами, на земній кулі міститься множина різних пристроїв з різною потужністю обчислення, на яких і вирішуються завдання користувачів.

Метою роботи є розробка ТОС, що пропонує ОР на базі концепції ТО, яка повинна контролювати, налаштовувати та надавати туманні ОР в автоматичному режимі.

Для досягнення мети потрібно вирішити наступні завдання:

– провести огляд аналітичних джерел за тематикою роботи та наявного ПЗ

для керування ТО;

– проаналізувати технології, котрі необхідні для втілення ТОС;
– спроектувати архітектуру ТОС;
– реалізувати ТОС;
– провести тестування реалізованої системи.

Об’єкт дослідження: концепція ТО та їх використання у програмних продуктах для надання ОР.

Предмет дослідження: програмний засіб надання туманних ОР.

Наукова новизна роботи: запропоновано прототип ТСО Fog_Comp для надання ОР.

Практичне значення одержаних результатів. Розроблений програмний продукт дає змогу успішно надавати ОР туману. Це є ефективним моментом для використання у тих сферах, у котрих є потреба роботи із системами , в яких є надзвичайно важливою їх швидка реакція на настання певних подій, для прикладу проведення аналізу у живому режимі.

Апробація. Окремі результати дослідження доповідалися на XI науково-технічній конференції «Інформаційні моделі, системи та технології» як опубліковані тези.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Концепція туманних обчислень

Більше десяти років назад виникла потреба удосконалити ХО, щоб дати раду із зростаючою множиною IoT- пристроїв та постійно зростаючими об’ємами даних, котрі постають при роботі, а також із додатками, котрі працюють в реалі і є надзвичайно чутливими до часових затримок. Варіантом розв’язання таких проблем і стали ТО, які зараз досить широко застосовуються через компанію Cisco, що розглядає їх як нову технологію, котра дає змогу організувати додатковий рівень взаємозв’язку між фінальними пристроями та хмарними ЦОД [1].

1.1.1 Основні дефініції

ТО є багаторівневою моделлю, що надає всеосяжний доступ до спільного континууму ОР, що масштабуються. ТО максимально мінімізують час відгуку ПЗ і пропонують для КП локальні ОР та, при необхідності, мережне під’єднання до служб Інтернету [2].

Поняттям ТО перебуває у тісному зв’язку із поняттям кордонних обчислень. До них належать механізми, котрі дають змогу поводити розрахунки на кордоні/межі мережі. Під "кордоном" тут маються на увазі будь-які ОР чи мережі між джерелами і хмарними ЦОД [3]. Логіка кордонних обчислень у тому, що вони повинні проходити біля джерел даних. З погляду авторів [3], такі обчислення взаємозамінні з ТО, але кордонні обчислення більше орієнтовані на речі, а туманні - на інфраструктуру.

1.1.2 Елементи туманних обчислень

ВТ є головним компонентом будови ТО. Вони володіють своїм географічним місцем розміщення і надають визначений вид керування послугами і даними між прикордонним рівнем мережі, власне в на якому і містяться КП, та службою ТО чи централізованими ОР, якщо в цьому є потреба [2].

Інтерфейси служби ТО – туман-хмара та туман-туман. На рис. 1.1 наведені ТО у ширшому контексті для обслуговування фінальних пристроїв. ТО не є обов'язковими для систем такого виду. Навіть, властиво, централізована служба також вважається потрібним рівнем ТО. Різноманітні скрипти застосування можуть володіти різною архітектурою, котрі ґрунтуються на оптимальному варіанті, спрямованому на підтримання дієвості КП [2, 4].

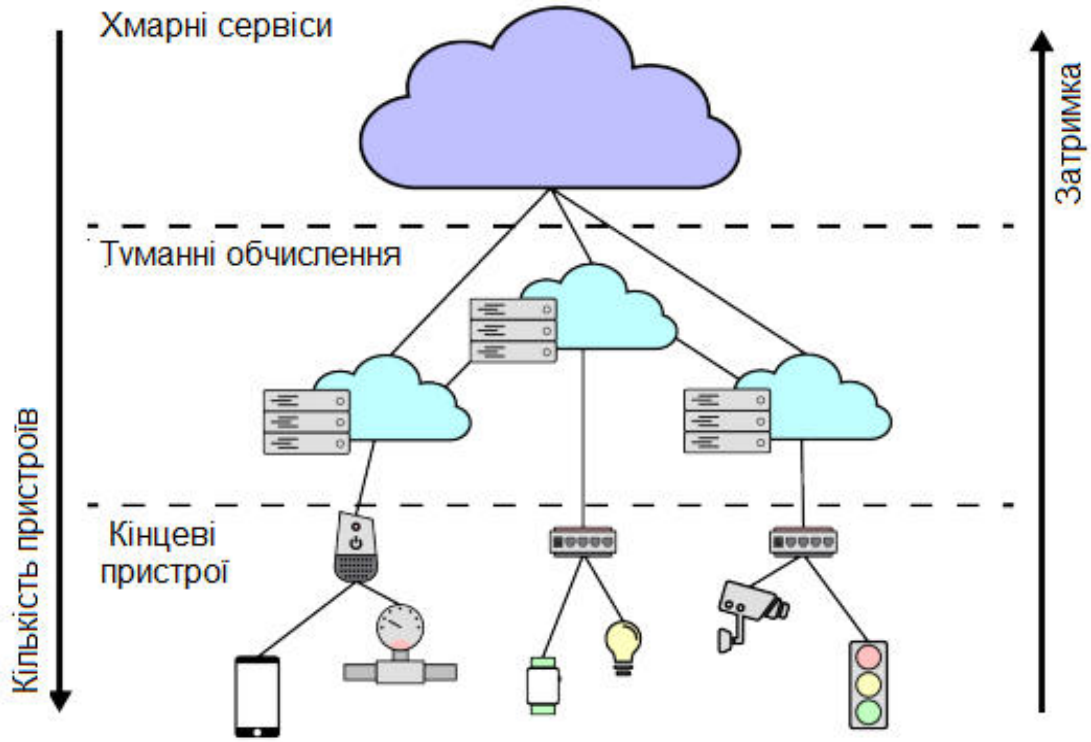

Рисунок 1.1 – Концепція ТО як підтримання інфраструктури в хмарі

1.1.3 Специфіка туманних обчислень

Фірми, котрі пропонують ОР на базі концепції ХО, не мають змоги забезпечити малу латентність їх систем, тому що ОР можуть знаходитися дуже далеко від КП. Цю проблему не можна розв'язати елементарним зростанням числа ЦОД фірм у всьому світі [5]. Проте для її розв'язання варто застосувати концепцію ТО, котра дає змогу [6, 7]:

– до мінімуму зменшити латентність між КП та ТОС;

– функціонувати в межах чинного нормативного законодавства різних країн щодо збереження ті опрацювання персональних даних юзерів;

– покращити стійкість системи до відмов завдяки втіленню децентралізованого опрацювання даних;

– забезпечити оптимізацію трафіку через магістральні канали, та зменшити завантаженість мережевого устаткування провайдерів через забезпечення опрацювання інформації на ПТ, котрі розміщені поруч із юзером.

Хоча ТО достатньо різняться від ХО, ТО не є їх повним замінником. Для розв'язання окремих задач ТО може бути не досить ОР, але є значною перевагою їх розміщення біля користувача. ТО повинні справитися із деякими недоліками, котрі мають ХО, та вдало доповнити їх. Застосування ТОС разом із системами на основі ХО дасть змогу побудувати цілісне рішення для забезпечення потреб кінцевого користувача.

1.1.4 Погляди до формування структури туманних обчислень

Переважна кількість існуючих рішень ТОС засновані на фундаментальній архітектурі трьох рівнів. Мається на увазі, що між IoT- пристроями та хмарою наявний рівень туману, котрий є проміжним. Окремі задачі втілюються локально,

як то попереднє опрацювання, очистка даних і їх фільтрація. Тільки якась частка даних, що є корисними, переходить до хмари [8].

Автори в [9] пропонують модель p2p туману. Вона покращує ТО, додаючи механізм p2p у шар туману, це дає змогу ВТ перебувати у постійному взаємозв’язку між собою. Тобто, автори мінімізують запити, що надходять в хмару, і виконують більшість запитів властиво саме на ВТ.

У [10] представили архітектуру представлення послуг мобільних ХО, що базується на ТО. Тут наявні точки доступу в мережі та блоки контролерів точок доступу як вузли мобільного туману, де вдосконалений шлюз даних є відповідальним усередині туману за зв'язок. Запропонована конструкція при якій точка доступу не лише забезпечує стабільний зв’язок між мобільним устаткуванням та IP -мережами, але і надає достатні можливості для функціонування додатків.

1.2 Механізми підтримки туманних обчислень

Організація ТО є нетривіальним завданням, особливо якщо вона має відбуватися автоматично. Якнайменше для її втілення потрібно відшукати такі ВТ, які є найближчими до місця вимоги ОР, встановити на них необхідні додатки, забезпечити стеження за ними та підтримку їх коректного функціонування. Для забезпечення всього, наведеного вище, повинна бути складна система, котра містить множину компонентів, котрі перебувають у взаємодії один з одним [11].

Зараз наявна значна кількість різноманітних програмних інструментів, при використанні яких обчислювальна система буде так чи інакше забезпечувати виконання тих завдань, які встановлені перед нею.

Для роботи різноманітних додатків на ВТ можна застосовувати відкрите ПЗ для розроблення, доставляння та завантаження програм – Docker [12]. Він дає змогу пакувати та виконувати програми у контейнерах, котрі дозволяють відділити

програми власне від інфраструктури, яка їх оточує.

Застосування критеріїв ізоляції та безпеки дають змогу виконувати разом на одному хості декілька контейнерів. Контейнери легковагові, оскільки вони не потребують гіпервізора, виконуються безпосередньо на ядрі.

Виконання контейнерів та їх підтримування у такому вигляді не можна проводити вручну, оскільки їх чисельність стає аномальною. Docker Swarm [13] якраз і призначений, щоб автоматизувати такі процеси. Це ПЗ надає множину засобів для масштабування, формування мереж, захисту та підтримування додатків у контейнерах, на додачу до властивостей контейнерів як таких. Цей засіб вмонтований у Docker Engine по дефолту і є досить простим у розумінні.

Для тонкого налаштування та керування контейнерами застосовують комплексні інструменти, з них найбільш поширеним є Kubernetes [14]. Застосовується для оркестрації контейнерів для забезпечення автоматизованого процесу розгортання, масштабування та керування застосунками на базі контейнерів. Для функціонування Kubernetes потрібно задати стан кластера, а саме вказати програми, їх чисельність для запуску, образи яких контейнерів застосовувати тощо. Після цього інструмент забезпечуватиме оптимальну підтримку кластера у визначеному стані. Платформа управляє не окремими контейнерами, а їх зв’язаними наборами, котрі носять назву Pod. Вони є величиною планування в Kubernetes. В цьому полягає відмінність від Docker Swarm.

Для розроблюваної ТОС функціональності Docker Swarm, буде достатньо, тому як основний засіб для оркестрації обраний саме він. З метою доставляння та виконання різних додатків на необхідних ВТ, Docker Swarm потребує сховища, у котрому міститимуться всі потрібні образи додатків. Тут зручно використати Docker Registry [15], котрий є масштабованим open-source серверним застосунком для зберігання і розповсюдження Docker- образів. На його базі функціонує Docker Hub, який є найбільшим у світі репозиторієм контейнерних образів.

1.3 Процедури забезпечення комунікації сервісів туману

При використанні мікросервісної архітектури важливим є забезпечення ефективної співпраці внутрішніх служб між собою. Наявна множина підходів та інструментів, котрі поділяються на синхронні і власне асинхронні.

До перших можна віднести два популярні рішення REST та RPC. REST [16] є сукупністю узгоджених вимог, направлена на співпрацю складових частин в мережі розподіленого застосунку, котрий, як правило, застосовується для створення Web-сервісів. Хоча REST не є стандартом сам по собі, більшість його реалізацій застосовують такі стандарти як HTTP і JSON.

Альтернативним механізмом до REST є підхід RPC [17], котрий дає змогу додаткам викликати програмні одиниці в чужому просторі адрес, прикладом може бути віддалений хост. Втілення механізму RPC містить дві складові: протокол мережі для обмінювання даними і власне мову серіалізації визначених об'єктів. Варто звернути увагу на gRPC [18], що є успішним втіленням згаданого підходу. Є надійним відкритим фреймворком з реалізацією RPC, забезпечує ефективну комунікацію між службами всередині та між ЦОД. Його переваги: застосування другої версії HTTP та Protocol Buffers [19], що робить час, потрібний для комунікації служб, значно меншим.

З метою забезпечення асинхронності для комунікації служб найчастіше застосовують черги повідомлень або, так звану, потокову обробку. Повідомлення перебувають в черзі на зберіганні, аж поки вони не будуть опрацьовані і потім знищені. Черги [20] можуть використовуватися для розділення комплексних процесів опрацювання, буферизації чи забезпечення пакетного опрацювання, а також для того, щоб зробити пікові навантаження більш згладженими. Обмінювання повідомленнями по традиції відбувається із застосуванням однієї з двох моделей: організація черг та публікація-підписка. При першій моделі користувачі можуть зчитувати дані із сервера, варто зауважити, що кожен запис буде переслано одному користувачеві; при другій - запис надсилається усім

користувачам, котрі володіють підпискою на чергу. Найбільшою популярністю у фахівців зараз користується RabbitMQ [21]. Він є найбільш поширеним відкритим брокером повідомлень, котрий гарантує доставляння повідомлень, володіє ефективною системою маршрутизації і оперує кількома протоколами обміну.

Потокова обробка [22] забезпечує здійснення операцій над набором даних, котрі генеруються повсякчас, при їх формуванні чи одержанні. Apache Kafka [23] є доступним розподіленим потоковим засобом, який при допомозі концепції ряду користувачів підсумовує обидві концепції обміну. Аналогічно до черги, ряд користувачів дає змогу поділити обробку на комплекс процесів, як і у випадку публікації-підписки, платформа дає змогу надсилати повідомлення різним об’єднанням користувачів. Kafka володіє більшою сукупністю критеріїв підписки, аніж звичайний засіб обміну.

Звичайно, що наведені вище технології комунікації володіють перевагами і недоліками, і, у залежності, від поставлених задач можуть використовуватися різні механізми. Для реалізації ТОС оптимальним фреймворком для реалізації комунікації вибрано gRPC.

1.4 Огляд наявних засобів для керування туманних обчислень

Концепція ТО існує вже більше 10 років, за цей час з'явилися деяка кількість як безкоштовних (open source), так і платних комерційних засобів, розроблених на її базі. Варто провести короткий аналіз найбільш відомих на теперішній час програмних інструментів, котрі забезпечують доступ до ОР туману.

ioFog [24] - повна периферійна платформа для проведення обчислень, котра надає всі програмні складові, що є потрібними для побудови та виконання додатків на кордонному устаткуванні в територіально розподіленому підприємстві. ПЗ ioFog Agent абстрагує багатогранність і комплексність обладнання периферії. Керування та оркестровка мікросервісів периферії здійснюється із використанням

ioFog Controller і його ж допоміжного діапазону складових елементів. Платформа володіє open-source кодом і безкоштовно поширюється.

Sonm [25] - надає хмарні послуги на основі розподіленого апаратного забезпечення рівня клієнта, включаючи комп'ютер, обладнання для майнінгу та сервери. Дозволено здавати своє устаткування в оренду або застосовувати чиїсь ОР для своїх потреб. Sonm є варіантом ХО, розв'язанні проблеми значної її латентності не є ключовою задачею, невирішеною є проблема географічного розміщення ОР.

Difuon [26] – фактично є аналогічною системою до попередньо розгянутої. Вона дає змогу споживачеві без спеціальних скілів, інсталювати ПЗ на своє устаткування і бути задіяним в процесі обміну ОР та отримувати від цього певний дохід. Керування ресурсами туману та застосунками забезпечується застосуванням хмарного сервісу, котрий зв'язує тих , хто надає і використовує ОР.

AWS IoT GreenGrass [27] - це інструмент від Amazon, в якого наявне виконання функцій AWS Lambda і контейнерів Docker на обладнанні периферії. Ця властивість надає можливість цьому устаткуванню локально функціонувати з даними, котрі вони формують, та ще й синхронізувати їх. Іншими словами, дані, що генеруються, опрацьовуються на кордонному устаткуванні, а інформація, котра має бути збережена, надсилається до хмари. Щоб застосувати цей інструмент, користувач повинен інсталювати на своє устаткування спеціалізоване ПЗ та здійснити його реєстрацію в AWS.

Azure IoT Edge [28] - служба від Microsoft, за функціональністю є аналогом до описаної попередньо. Вона дає змогу застосовувати власну бізнес-логіку юзерів на кордонному IoT-обладнанні при використанні звичайних контейнерів, а для довгострокового збереження інформації призначена хмара. Споживач також повинен інсталювати ПЗ на свої пристрої та виконати їх реєстрацію у системі Azure.

Провівши огляд окремих інструментів для забезпечення можливості ТО можна прийти до висновку, що достатньо багато платформ розв'язують проблему значної латентності системи при опрацюванні даних, проте юзер має самостійно інсталювати спеціалізоване ПЗ на свої пристрої, здійснити його налаштування і

реєстрацію. Також переважно потрібна хмара для зберігання інформації, а це, знову ж таки, зростання часу доступу до неї. У ТОС, котра розробляється, буде передбачено автоматичне виділення і налаштування ОР, які географічно найближче розміщені до користувача.

1.5 Висновок до першого розділу

В цьому розділі проведено аналіз предметної області дослідження ТО, а саме: основні визначення, складові елементи, специфіка та особливості формування архітектури. Також розглянуто технології підтримання ТО та організації комунікації туманних програм. Наведено основні програмні інструменти для роботи з ТО.

2 ПРОЕКТУВАННЯ РІШЕННЯ

Ключовою ідеєю цього дослідження є побудова архітектури функціонування та втілення PaaS-рішення «Fog_Comp», котре надасть ефективний та коректний спосіб розгортання контейнерів та сервісів на їх основі відповідно до концепції ТО.

Для ТОС Fog_Comp було визначено такі два ключові критерії:

– застосування хмарних ОР для функціонування основних її компонентів, що дасть змогу управляти ПТ централізовано;

– застосування концепції ТО на додачу до технологій контейнеризації та оркестровки власне для децентралізованої вже реалізації задач юзерів.

2.1 Вимоги до системи

Нижче представлені ключові функціональні та нефункціональні вимоги для ТОС Fog_Comp, що розробляється.

2.1.1 Функціональні вимоги

ТОС має:

– виконувати розгортання застосунків на туманних ОР якнайбільш географічно ближче до IoT-обладнання;

– здійснювати розгортання додатків туману на вимогу IoT-устаткування автоматично;

– ґарантувати підтримку потрібних програм у статусі їх виконання навіть при аварійному закінченні їх функціонування чи при аварійному виключенні

устаткування туману;

– надавати можливість функціонування множині IoT - обладнання із варіантом програми, котра виконується;

– давати змогу функціонування одного пристрою з декількома туманними додатками.

2.1.2 Нефункціональні вимоги

Вони є такими:

– ПТ мають функціонувати на обладнанні під управлінням ОС Linux;

– ТОС має застосовувати ізольовані контейнери для реалізації програм для полегшення і спрощення процесу розгортання;

– система має давати нагоду виконувати теж саме ПЗ на різному обладнанні, додержуючись разом із тим усіх правил, котрі притаманні додаткам.

2.2 Варіанти використання

На рис. 2.1 відображена діаграма прецедентів ТОС. Система повинна забезпечувати одночасну функціональність з 4 класами користувачів: адміністратор, розробник, IoT -пристрій та IoT Hub.

Адміністратор - актор, який володіє розширеними можливостями, скерованими на управління ТОС для підтримування її у стані працездатності. Функціональність цього актора:

- додати ПТ*:* актор інсталює потрібне ПЗ на ПТ та виконує його реєстрацію в системі;

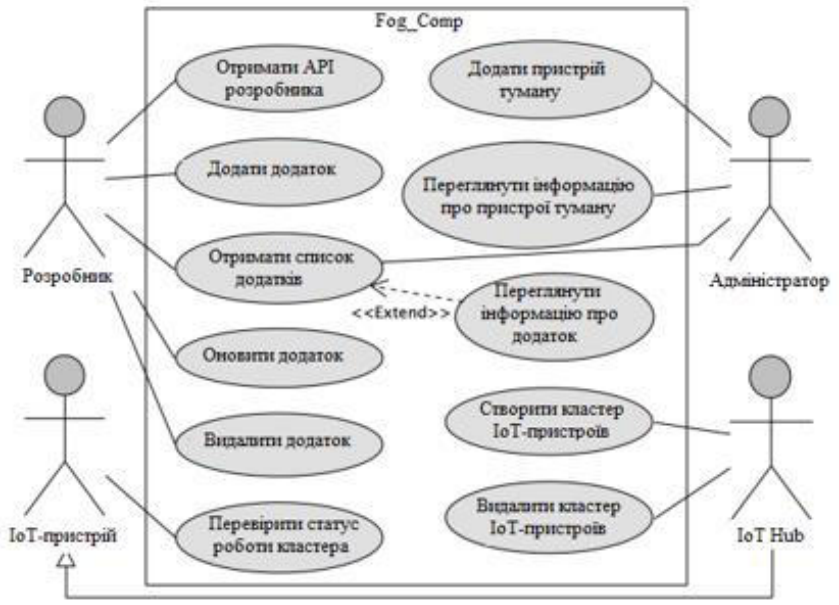

Рисунок 2.1 – Діаграма прецедентів системи Fog_Comp

- переглянути інформацію про ПТ: їх чисельність, технічні параметри і стан;
- одержати список додатків*:* актор може одержати перелік всіх наявних у системі застосунків, а також здійснити перехід до визначеного додатку, щоб одержати більш докладну інформацію про нього.

Розробник - є приватником чи установою, що потрібує туманні ОР для запуску їх туманних застосунків. Його можливості в межах ТОС, яка проектується:

- одержати API розробника: IoT-пристрою для взаємодії з ТОС повинні застосовувати методи API;
- додати додаток*:* розробники мають змогу поміщати образи застосунків до ТОС, котрі за потреби будуть вподальшому розгорнуті на ПТ;
- оновити додаток*:* за потреби внесення змін чи довнесення оновлених функцій до застосунку, розробник здатен завантажити оновлений варіант образу додатку;
- видалити додаток, за умови, що він більше не є потрібним;
- отримати список додатків: перелік програм, які розробник завантажив у систему, може ним передивлятися, також він здатен виконати перехід перейти до

визначеного додатку, щоб одержати більш докладну інформацію про нього.

IoT-пристрій – є якимось автономним пристроєм, котрий під'єднаний до інтернету, який потребує туманні ОР. Множина такого обладнання, що працюють з одним туманним додатком формують кластер такого устаткування.

IoT Hub - є центральним IoT- пристроєм, який дає змогу решті устаткування під'єднатися до власне кластера. Цей хаб володіє усіма можливостями IoT-обладнання, а також може формувати їх кластер, надавши свої географічні координати ТОС Fog_Comp, яка відшукає самий ближчий до хаба ВТ і виконає на ньому потрібну програму в контейнері. Такий хаб також має можливість видалення сформованого раніше кластера IoT -пристроїв, за умови, що його послуги більше не є потрібними.

Після формування кластеру пристроїв, IoT -пристрій здатен здійснити перевірку статусу його функціонування, щоб зрозуміти на якому власне ПТ виконується потрібний додаток. Коли Fog_Comp фіналізує розгортання додатку, IoT -пристрій стартує для роботи із тим екземпляром туманного додатку, котрий був попередньо запущений.

2.3 Архітектура системи

Ядро архітектури ТОС складається з наступних компонентів: API Gateway, Users Service, Cloudlets Service, Images Service, Scheduling Service, Web Application. Ядро управляє під'єднаними ПТ, контролює їх стан, забезпечує планування, розподіл і виконання користувацьких додатків. На додаток до ядра, існує набір децентралізованих ПТ, на яких власне і проводиться виконання тих завдань, котрі поставлені користувачем. Для виконання програм користувача на всіх ВТ має бути розгорнута стандартне ПЗ, котре виконує контейнери із образами програм туману.

Для програмного втілення ядра платформи була обрана мікросервісна

архітектура, котра складається із 7 сервісів та ПЗ, що функціонує на ВТ (див. рис. 2.2). Кожен сервіс є елементом, котрий функціонує в контейнері, який є ізольованим. Застосування такого виду структури дає змогу розкласти велику систему на множину малих і, відповідно, простіших сервісів, котрі перебувають у взаємодії між собою. Мікросервісна конструкція надає перспективу масштабування окремо будь-якого сервісу для забезпечення балансування навантаження, якщо така потреба виникне.

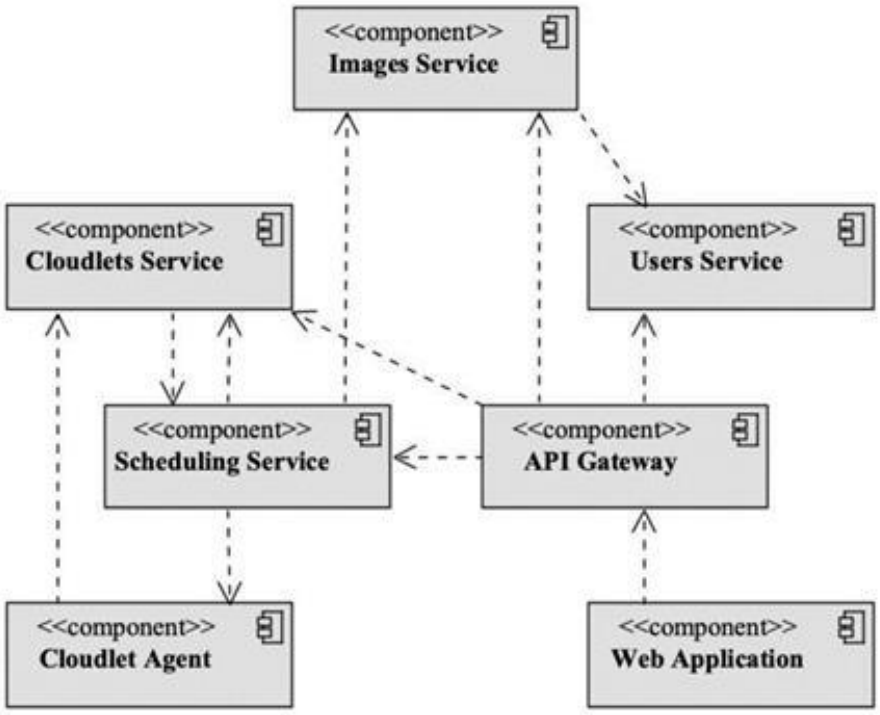

Рисунок 2.2 – Діаграма компонентів ТОС Fog_Comp

API Gateway є головною точкою входу до ТОС. Будь-який одержаний запит автоматично буде перенаправлений до потрібних служб, як тільки він виконається – відразу буде сформована визначена відповідь, котра і повернеться до користувача. Цей механізм є досить поширеним при проектуванні мікросервісних систем і володіє рядом плюсів. Одним з найбільш значущих є гнучкість такої програми: внутрішня її будова може піддаватися зміні необхідну кількість разів, проте це не вимагає змінювати інтерфейс взаємодії з програмою [29].

Users Service є службою, котра несе відповідальність за зберігання та опрацювання даних про системних юзерів, а також пропонує методи, які дозволять авторизувати наявних та провести реєстрацію нових юзерів.

Cloudlets Service - призначенио для збереження та опрацювання даних щодо всіх ПТ, під'єднаних до системи. Служба містить такі дані, як географічне положення обладнання, його технічні параметри, версії ПЗ і т.п. Вона ще і пропонує метод для створення нових ПТ і їх додавання та метод відшукання за географічними координатами пристроїв, котрі найближче мітяться до визначених.

Images Service забезпечує доступ до сховища, в якому зберігаються образи програм туману. Кожен юзер для коректної роботи з ТОС поміщає образи свого ПЗ в сховище. Вони будуть виконані в ізольованих контейнерах на ПТ.

Scheduling Service є службою, котра несе відповідальність за план і втілення задач користувача на ВТ. Вона здійснює опрацювання усіх запитів, котрі поступили на надання доступу до ОР туману, при допомозі Cloudlets Service шукає обладнання, котре розміщається поряд з IoT- пристроями і виконує потрібні програми в контейнерах, які є ізольованими.

Web Application - це додаток, який розташований у хмарі, проте взаємодіє успішно з іншими програмами з використанням API Gateway. Головною ідеєю застосування цього веб-додатку є забезпечення зручної можливості для ефективної взаємодії програмістів та адміністраторів з ТОС через будь-який бравзер.

Cloudlet Agent - це ПЗ, інстальоване на всіх ВТ, призначене для управління із хмари ПТ, поєднує ресурси обладнання у платформу, забезпечуючи розгортання обчислювальних можливостей.

2.4 Додавання образу додатку

Так як ТОС Fog_Comp має пропонувати не тільки доступ до ОР, які знаходяться поряд, а ще й розгортати потрібні програми, які вимагає IoT –пристрій,

на цих ресурсах, тому потрібно, щоб фахівець виконав підготовку свого додатку та доповнив ним систему. Черговість необхідних дій, яка має бути проведена програмістом для додавання додатку, зображена на рис. 2.3.

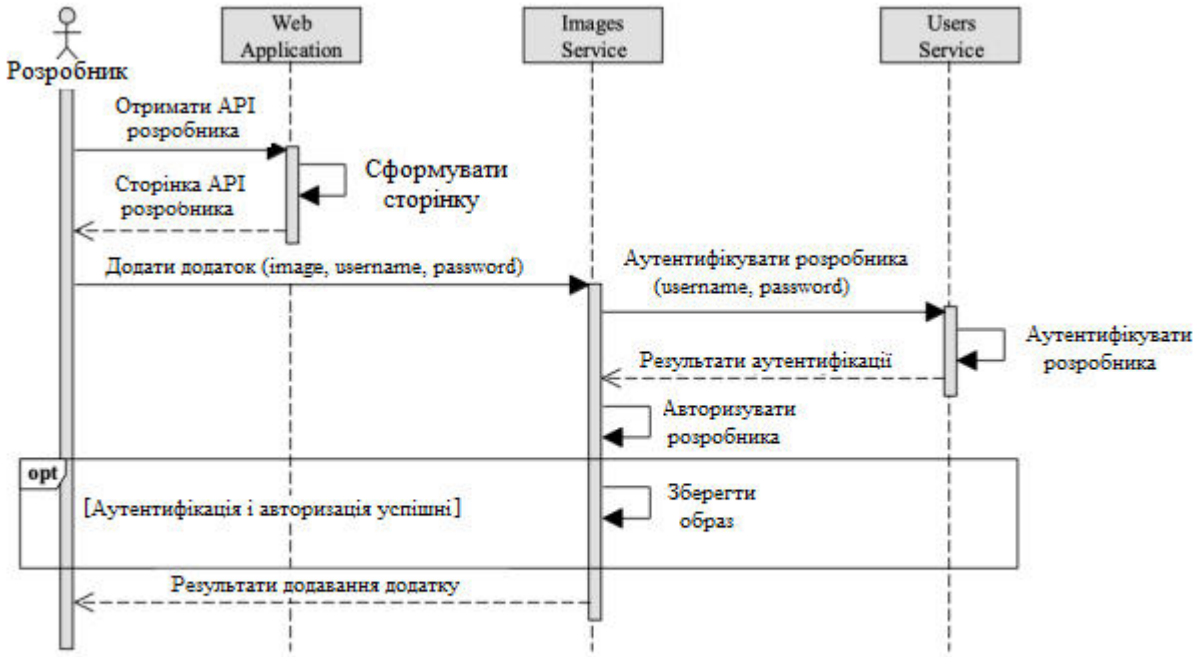

Рисунок 2.3 – Діаграма послідовності додавання образу програми у систему Fog_Comp

Перш за все фахівець має одержати API розробника, згідно з яким він і створить програму. З цією метою він при допомозі Web бравзера запитує Web Application, що робить та вертає HTML документ з описом методів API та потрібних умов до програм. Процес самої розробки додатку не показано на рис. 2.3, так як це є задачею програміста і він має самостійно прийняти рішення, що за технології застосовувати та яку практичність і надійність втілювати. Проте є одна чітка вимога - в ТОС треба помістити Docker образ додатку.

Коли у програміста є сформований образ додатку, тоді з використанням Docker Client він доповнює ним Docker Registry, котрий є складовим елементом Images Service. Той при допомозі Users Service автентифікує розробника, іншими словами здійснює перевірку, що в ТОС дійсно існує власне розробник з точно таким іменем користувача і паролем, котрий їм надається, і ці дані є правильними.

Після аутентифікації проходить авторизація програміста: Images Service здійснює перевірку того, чи він працює з належним йому сховищем. Це для того, щоб програмісти не мали змоги одержати доступ до образів програм, на які вони не мають права. Якщо обидва ці процеси успішно завершилися, тоді образ додатку буде збережений зберігається в ТОС. Кінець кінцем розробник одержує результат довнесення програми, і не має значення той факт чи буде він успішним чи навпаки.

2.5 Додавання пристроїв туману

Для того, щоб ТОС Fog_Comp могла успішно і ефективно здійснювати розгортання образів програм, потрібно, щоб у неї були туманні ОР. Кожен ПТ є обчислювальним обладнанням під ОС Linux, на котрому має бути розгорнуто базове ПЗ. Елементом його є Cloudlet Agent, котрий призначений для управління пристроєм з хмари і дає змогу провести реєстрацію нових туманних ОР в системі. Порядок проведення дій, який проводить Cloudlet Agent під час реєстрації, показано на рис. 2.4.

Перш за все Cloudlet Agent виконує перевірку даних щодо успішного реєстрування обладнання, які містяться у файлі локальної ФС ПТ. Якщо ПТ вже зареєстрований у ТОС, агент продовжує роботу у режим фону для здійснення інших дій. Якщо ПТ не зареєстрований, агент проводить збір необхідних даних про нього: технічні параметри, географічні координати, IP адреса, ОС тощо.

Після завершення отримання даних про ПТ Cloudlet Agent при допомозі адміністратора здійснює їх редагування та підтвердження їхньої правильності, аж поки вони не стануть коректними. Якщо все правильно, агент використовує віддалену реєстрацію для ПТ Cloudlets Service. Залежно від того, чим вона закінчилася, агент надає адміністратору висновок. Якщо реєстрація ПТ успішна, він зберігає дані про неї у файлі, потім робота його продовжується.

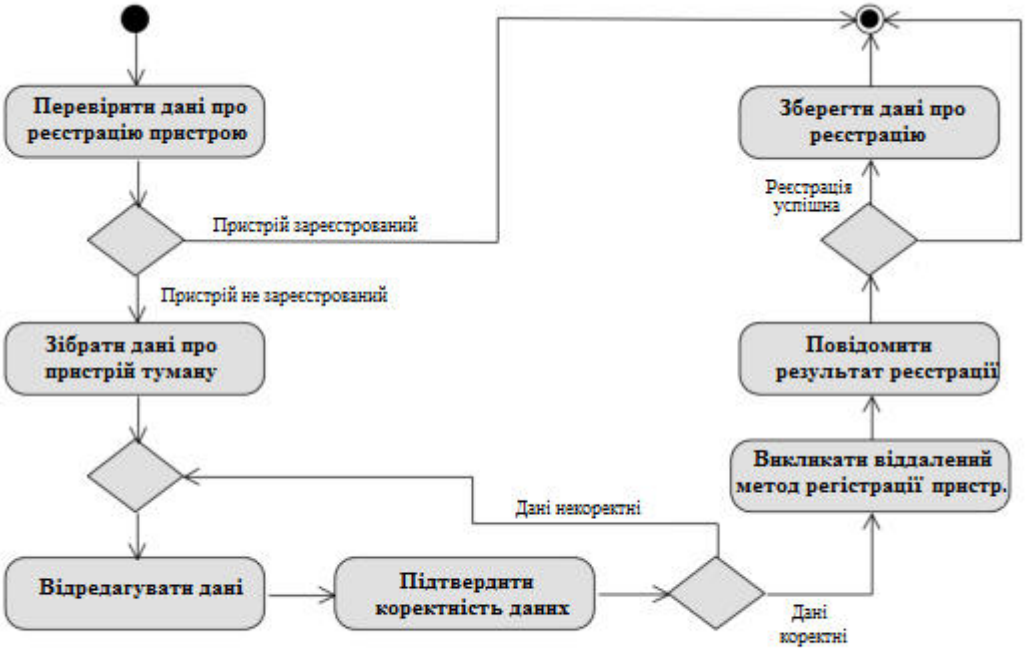

Рисунок 2.4 – Діаграма активності Cloudlet Agent при додаванні нового ПТ

2.6 Запуск туманної програми

Після того як адміністратор додав ВТ, а програмісти виконали завантаження образів своїх програм, ТОС Fog_Comp перебуває у стані готовності до їх виконання. Послідовність виконання туманних програм показано на рис. 2.5.

Для того щоб ТОС Fog_Comp почала розгортання додатку на ВТ, IoT Hub потрібно відправити повідомлення API Gateway для побудови кластера IoT - обладнання, Запит має містити географічне місце розміщення та ім’я образу, котрий потрібний. API Gateway, одержавши його, перенаправить його до Scheduling Service, котрий, і собі, повинен переслати його до Images Service для одержання потрібного образу додатку туману. Images Service відшуковуватиме образ із запиту у ТОС і верне його Scheduling Service. Потім цей сервіс відсилає повідомлення до Cloudlets Service для відшукання ПТ, які є біля хабу. Cloudlets Service із використанням координат відшукає ті ВТ, які будуть найближчими до IoT Hub, і надасть планувальнику дані про них.

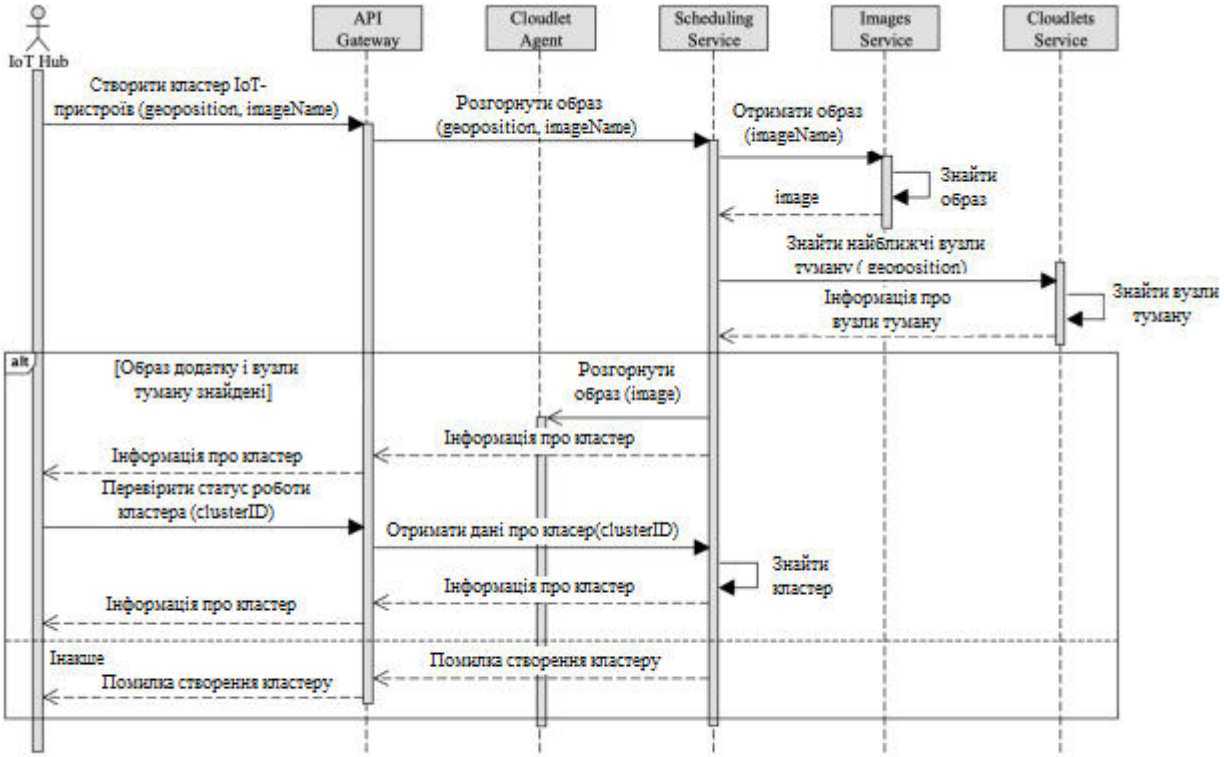

Рисунок 2.5 – Діаграма послідовності запуску туманних додатків

При умові, що взірець необхідного додатку та ВТ виявлені, то Scheduling Service, маючи усю потрібну інформацію, при допомозі Docker Swarm стартує для розгортання одержаного образу додатку на якомусь з ПТ. Так як завантаження додатку потребує певного терміну, він проходить в асинхронному режимі. Сервіс, не очікуючи завершення виконання додатку, повертає API Gateway дані щодо тільки що сформованого кластеру, котрий, зі свого боку, вертає її хабу назад як результат запиту про формування кластера. Як тільки IoT Hub одержує інформацію щодо кластера, котра містить його ID, він здатен виконати перевірку ступеня функціонування кластера, відправивши свіжий запит до API Gateway, для того щоб довідатися про закінчення перебігу розгортання додатку та одержати дані про ПТ, на котрому його буде. API Gateway вкотре здійснює перенаправлення цього запиту до Scheduling Service, котрий відшукує кластер за ID і виконує повернення інформації про нього у зворотньому напрямку. За умови, що IoT Hub цікавиться програмою, котрої не має у системі чи у системі відсутні ВТ, то Scheduling Service не здатен сформувати кластер, і він надсилає ваду.

2.7 Під’єднання до програми туману

Коли хаб сформував кластер IoT -устаткування, водночас запустивши потрібний образ на ПТ, котрий є найближчим до нього, інше IoT- обладнання аналогічно можуть застосвувати цей екземпляр працюючого додатку та виконати під’єднання до кластера.

Тому, щоб IoT -пристрій зумів під’єднатися до кластера, він повинен взнати його ID, котрий має надати IoT Hub. Перебіг передання даних щодо ID кластера відсутній на представлених діаграмах, оскільки він втілюється розробником додатка і не належить до ТОС Fog_Comp. Коли IoT –обладнання таки отримає ID потрібного кластера, він здатен відправити повідомлення API Gateway з метою отримання інформації про нього. З’ясувавши на котрому ВТ додаток є розгорнутим, IoT- обладнання закінчує роботу з API Gateway і стартує взаємодія з придатним до використання екземпляром додатку. Перебіг опрацювання з програмою знову є таки не показаний на діаграмах, так як кожна програма володіє різним функціоналом і застосовує різні механізми для взаємодії. ТОС Fog_Comp не слідкує за тим, які власне пристрої в які властиво кластерах містяться, та як проходить їх взаємодія з програмами, її задача тільки забезпечення доступу до програм на тих ВТ, які є найближчими.

2.8 Висновок до другого розділу

Було описано функціональні та нефункціональні вимоги до ТОС Fog_Comp, проведено її проектування. Встановлені основні актори, які можуть взаємодіяти із Fog_Comp та визначені їхні можливості. Спроектовано усі компоненти системи, для кожного з них зазначено поле його діяльності та практичність, котру він має провадити. На додачу було описано основні варіанти застосування ТОС.

3 ПРАКТИЧНА ЧАСТИНА

Будуть розглянуті компоненти втілення складових частин ТОС Fog_Comp, описуються методи застосування обраних програмних інструментів і технологій. Основною мовою розробки було обрано Python [30], а як засіб для збереження NoSQL даних - БД MongoDB [31].

3.1 Додавання образу програми

Для втілення процесу збереження образів туманних програм у ТОС Fog_Comp послуговуєтьс Docker Registry, фрагментарно вміст файлу із параметрами його налаштуваннями наведений на рис. 3.1.

Старт роботи Docker Registry виконується з консолі, такт як це контейнер, але щоб його налаштувати потрібно скласти yaml файл, котрий матиме усі необхідні атрибути. Одним з важливих є rootdirectory, котрий є відповідальним за місце збереження образів програм. Якщо навантаження на сховище відсутнє, його не треба масштабувати, саме тому зберігати програми можна і у ФС контейнера. Для початку роботи Docker Registry треба також задати адресу і порт, з яким му він буде взаємодіяти із використанням атрибуту addr.

Також потрібно провести налаштування надсилання повідомлень, якщо траплятимуться певні події. Для цього використовується розділ notifications, де при застосуванні атрибуту url розміщається службова адреса, котра прийматиме такі повідомлення. Атрибут ignore призначений для виключення лишніх типів подій та даних.

```
version: 0.1

storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /var/lib/registry
  delete:
    enabled: true

http:
  addr: :5000
  headers:
    X-Content-Type-Options: [nosniff]

notifications:
  events:
    includereferences: false
  endpoints:
    - name: ImagesService
      disabled: false
      url: http://<images_service_ip>:82/event
      timeout: 500ms
      threshold: 5
      backoff: 1s
      ignoredmediatypes:
        - application/octet-stream
      ignore:
        mediatypes:
          - application/octet-stream
        actions:
          - pull

auth:
  token:
    realm: http://<images_service_ip>:82/auth
    service: "registry.docker.com"
    issuer: "auth.docker.com"
    rootcertbundle: /mnt/local/certs/token.crt
```

Рисунок 3.1 – Частина файлу налаштувань Docker Registry

Щоб автентифікувати чи авторизувати юзерів при опрацюванні даних з репозиторію застосовується функція auth (рис. 3.2), у котрій потрібно задати вид авторизації, в цьому конкретному випадку ТД, та задати адресу тієї служби, котра буде їх видавати. Їх застосування є найкращим рішенням у ТОС, тут для перевірки автентичності користувацького ім’я та його пароля застосовується Users Service. При застосуванні як способу авторизації ТД потрібно також виконати генерування TSL сертифікату для Docker Registry, open складова котрого потрібна для підпису ТД, і, при допомозі атрибуту rootcertbundle, задати шлях до нього.

```
def auth():
  account = request.values.get('account')
  service = request.values.get('service')
  scopes = request.values.getlist('scope')
  if account:
    response = users_service_stub.VerifyUser(images_ser-
vice_pb2.User(username=request.authorization.username,password=re-
quest.authorization.password))

    if response.status.code == 200:
      access = []
      for scope in scopes:
        scope_dict = scope.split(':')
        if scope_dict[1].startswith(request.authorization.username +
'/'):
          access.append({
              'type': scope_dict[0],
              'name': scope_dict[1],
              'actions': scope_dict[2].split(',')})
      access_token = jwt.encode({'iss': 'auth.docker.com',
                                 'sub': request.authorization.username,
                                 'aud': service,
                                 'iat': datetime.datetime.utcnow(),
                                 'nbf': datetime.datetime.utcnow(),
                                 'exp': datetime.datetime.utcnow() +
datetime.timedelta(seconds=60),
                                 'jwi': str(uuid.uuid4()),
                                 'access': access},
                                app.config['PRIVATE_KEY'],
                                algorithm='RS256',
                                headers={'kid': app.config['KID']}).de-
code('utf-8')
      return {'token': access_token}, 200

  return '', 401
```

Рисунок 3.2 – Секція auth для проведення процесу автентифікації та авторизації

При запиті доступу до будь-якого додатку, Docker Registry відправляє повідомлення авторизаційному сервісу, в котрому при допомозі заголовку scope надає сховища, до котрих юзер хоче отримати доступ, а із використанням основної авторизації – його ім'я і пароль. З метою їх аудиту Images Service посилає вимогу до Users Service, якщо ці дані не є ідентичними, Власне Images Service відправляє назад відповідь з кодом 401, котра інформує Docker Registry, що аутентифікація юзера була невдала. Окрім згаданих параметрів Images Service також проводить перевірку атрибутів доступу юзера до сховища даних: користувач може взаємодіяти лише з образами тих програм, котрі містяться у його репозиторії. При успішній аутентифікації юзера формується ТД, у котрому містяться образи додатків, які успішно авторизувалися, потім він надсилається назад Docker Registry з HTTP статусом 200, що свідчить про успішність перевірки.

3.2 Додавання пристроїв туману

Для того, щоб ТОС Fog_Comp змогла дати IoT -обладнанню доступ до додатків, адміністратор має додати ПТ, на яких будуть запускатися всі потрібні додатки. Щоб зареєструвати новий ПТ Cloudlet Agent має відібрати потрібну інформацію щодо ВТ і відправити її до Cloudlets Service, котрий виконає її збереження і поверне ВТ його ID.

Агент, після свого запуску, передусім має перевірити чи конкретний ПТ є зареєстрованим, якщо ж це не так, то він повинен виконати його реєстрацію. З цією метою Cloudlet Agent виявляє об'єкт у локальній ФС, де у JSON- форматі мають бути збережені дані про реєстрацію ПТ. Якщо цей файл наявний, агент робить спроби продивитися його вміст і повинен «перекласти» його з формату JSON на мову Python. Якщо помилки трапляються чи файлу немає Cloudlet Agent запускає код реєстрації ВТ в ТОС, у всіх інших випадках його робота продовжується у фоновому режимі для реалізації інакших задач. Код, котрий призначений для реалізації описаних дій, подано на рис. 3.3.

```
if os.path.exists(cloudlet_info_file_path):
    with open(cloudlet_info_file_path, 'r') as file:
        try:
            cloudlet_info = json.loads(file.read())
        except json.decoder.JSONDecodeError:
            register_cloudlet()
else:
    register_cloudlet()
```

Рисунок 3.3 – Функція перевірки процесу реєстрації нового ПТ

Набір дій, котрий реалізує код реєстрування нового ПТ в ТОС, можна умовно представити у двох стадіях. На першій з них Cloudlet Agent при застосуванні стандартних вказівок консолі Linux і шаблонної бібліотеки Python subprocess нагромаджує інформацію про ПТ, де він запущений. Для прикладу, при допомозі hostname код реєстрування одержує неповторне ім'я хоста, вказівка lscpu надає інформацію щодо процесора і т.д. Вподальшому, застосовуючи функцію

необхідного типу, показану на рис. 3.4, агент надсилає просьбу адміністратору щодо зміни зібраних даних чи підтвердження їхної правильності. Цей код також показує на екрані машини інформацію про те, котрий атрибут в даний час піддається редагуванню та здійснює перевірку, що адміністратор вводить коректні дані. Описані вище дії повторюється до тих пір, аж поки дані потрібного типу будуть якісно одержані.

```
def required_type_input(message, old_value):
  while True:
    print(message, f'[{old_value}]: ', end='')
    new_value = ''
    while True:
      new_value = input()
      if old_value:
        break
      if not new_value:
        print('Empty value is not valid.')
        print(message, f'[{old_value}]: ', end='')
      else:
        break

    if not new_value:
      new_value = old_value

    try:
      return type(old_value)(new_value)
    except ValueError:
      print('Unexpected value.')
```

Рисунок 3.4 – Функція перевірки правильності типу введених даних

На другій стадії агент віддалено викликає функцію Add у Cloudlets Service та надсилає комплект зібраних даних для реєстрування ВТ. Після реалізації віддаленого коду стан роботи буде показаний для інформації на екрані. Здобувши цю інформацію, хостове ім'я міняється на ID устаткування, щоб менеджер зумів тотожно розпізнати цей ВТ, після чого його буде приєднано до рою. Якщо процедура додавання успішно закінчилася, одержана інформація про ВТ, який зареєстровано, зводяться до формату JSON і будуть збережені у локальному об’єкті. Програмний код, який втілює це, наведений на рис. 3.5.

```
stub = cloudlet_agent_pb2_grpc.CloudletsAPIStub(channel)
response = stub.Add(cloudlet_agent_pb2.Cloudlet(
                name=name,
                cpu_cores=cpu_cores,
                cpu_frequency=cpu_frequency,
                ram_size=ram_size,
                rom_size=rom_size,
                os=os,
                os_kernel=os_kernel,
                ip=ip,
                latitude=latitude,
                longitude=longitude,
                country=country,
                region=region,
                city=city))

print(f'\nResponse:\n\tCode: {response.status.code}\n\tMessage: {re-
sponse.status.message}\n')

if registration_response.status.code in [201, 409]:
  subprocess.run('hostnamectl set-hostname ' + registration_re-
sponse.cloudlet.id, shell=True)
  print('\nJoin to Swarm status: ')
  success = False
  try:
    docker_client.swarm.join(remote_addrs=[registration_re-
sponse.swarm_manager_address],join_token=registration_re-
sponse.swarm_worker_token)
    success = True
  except docker.errors.APIError as error:
    print(f'\t{error}')
    if 'This node is already part of a swarm' in str(error):
      success = True

  if success:
    print('\tSuccess\n')
    with open(cloudlet_info_file_path, 'w') as file:
      data = MessageToJson(registration_response.cloudlet)
      file.write(data)
      return json.loads(data)
```

Рисунок 3.5 – Код Cloudlet Agent для реєстрації ПТ

Коли Cloudlets Service отримує виклик методу Add щоб додати новий ПТ, він має виконати збереження даних про сайт у локальній БД. Для цього він спочатку зводить дані до свого стандартного типу diet та відправляє запит до Scheduling Service для отримання даних щодо менеджера рою. Після цього він пробує довнести новий ПТ у їх колекцію.

У cloudlets_collection поле ip повинно бути неповторним для кожного окремого ПТ, тому Cloudlets Service опрацьовує помилку. Коли при внесенні нового запису трапляється DuplicateKeyError, це свідчить про те, що ПТ з такою ж самою IP адресою вже наявний, тоді Cloudlets Service відшукує потрібну інформацію про нього у БД і віддає її назад як результат. Крім DuplicateKeyError Cloudlets Service аналогічно слідкує за будь-якими іншими помилками, щоби

правильно їх опрацювати і видати тескт з кодом 500, котра говорить про помилку всередині сервера (див. рис. 3.6).

```
new_cloudlet = MessageToDict(request, preserving_proto_field_name=True)

with grpc.insecure_channel(self.scheduling_service_url) as channel:
  stub = cloudlets_service_pb2_grpc.SchedulingAPIStub(channel)
  response = stub.SwarmManager(cloudlets_service_pb2.Empty())
  if response.status.code != 200:
    return cloudlets_service_pb2.ResponseWithCloudlet(status=re-
sponse.status)
  swarm_address = response.manager.address
  swarm_token = response.manager.join_worker_token

try:
  object_id = self.cloudlets_collection.insert_one(new_cloudlet).in-
serted_id
  self.areas_collection.update_one(
    {'_id': f'{int(request.latitude)}x{int(request.longitude)}'},
    {'$push': {'cloudlets': {
                'id': str(object_id),
                'latitude': request.latitude,
                'longitude': request.longitude}}},
    upsert=True)

except pymongo.errors.DuplicateKeyError:
  existing_cloudlet = self.cloudlets_collection.find_one({'ip': re-
quest.ip})
  existing_cloudlet_proto = cloudlets_service_pb2.Cloudlet()
  ParseDict(existing_cloudlet, existing_cloudlet_proto, ignore_un-
known_fields=True)
  existing_cloudlet_proto.id = str(existing_cloudlet['_id'])
  return cloudlets_service_pb2.ResponseWithCloudlet(
    status=cloudlets_service_pb2.Response(code=409, message='Cloudlet
with this IP address already exists.'),
    cloudlet=existing_cloudlet_proto,
    swarm_manager_address=swarm_address,
    swarm_worker_token=swarm_token)
except Exception as error:
  return cloudlets_service_pb2.ResponseWithCloudlet(status=cloudlets_ser-
vice_pb2.Response(code=500, message=f'An internal server error occurred.
{error}'))
```

Рисунок 3.6 – Частина функції Cloudlets Service для реєстрації ВТ

Якщо записування інформації про влаштування туману є успішним і не трапилося ні однієї помилки, тоді в areas collection до запису, що відповідає за область розміщення ПТ, додають ID ПТ та його координати (широта і довгота). Дубляж потрібен, щоб потім при відшуканні ПТ, котрі є найближчими до визначених координат, сервісу не потрібно було переглядати повністю усі ПТ.

3.3 Відшукання найближчих пристроїв туману

Як тільки ТОС Fog_Comp одержує запит для застосування туманних ОР, вона має, базуючись на географічних координатах, котрі фігурують у запиті, забезпечити визначення тих ВТ, котрі є найближчими до них і виконати на котромусь з ВТ потрібний додаток. Для від відшукання таких ВТ Cloudlets Service володіє методом FindNearest, котрий повертає ID’s двох ВТ, що є поряд з координатами, які одержані.

Щоб ресурси процесора були якомога ефективніше використані та часові витрати витрачати на відшукання найближчих ПТ були якомога меншими, у БД Cloudlets Service наявний метод araes, в котрий довносяться дані під час доповнення нових ПТ і який має змогу зберігати дані про те, в якій саме області ПТ перебувають. Поняттям «область» в ТОС Fog_Comp визначається ділянка земної поверхі з заданими географічно її цілими координатами. Наприклад, якщо ПТ перебуває на 48,1027° пн шир. і 23,1723° сх довг., тоді він попадає у область від 48 до 49° північної широти і від 23 до 24° у східній довготі, в ТОС така ділянка має ID «48x23» - Хуст, Ужгород.

Код відшукання найближчих ПТ показано на рис. 3.7. Для початку відшукання одержати перелік усіх областей, в яких є хоча б один ПТ. Одержавши його із БД, Cloudlets Service вносить дані в словник dist to areas. Для цього обчислюється дистанція від області, де треба ОР, до всіх областей з існуючими ПТ, одержана величина береться як ключ, щоб записати потрібні області. Заповнивши словник, його сортують за зростанням власне ключів.

Cloudlets Service після одержання відсортованих областей з ПТ, кладе початок пошуку ПТ у тих областях, які географічно біля нього. З цією метою він відшукує у БД перелік ВТ, у котрому міститься їх ID та координати. Для будь-якої з областей аналогічним чином, вираховуючи дистанцію до кожного ВТ, формується довідник disttocloudlets.

```
areas_id = self.areas_collection.find_one({'_id':'areas_id'}).get('list')
dist_to_areas = {}
for area_id in areas_id:
  area_latitude, area_longitude = area_id.split('x')
  area = Point(latitude=int(area_latitude), longitude=int(area_longi-
tude))
  dist = calculate_dist(search_area.latitude, search_area.longitude,
area.latitude, area.longitude)
  if dist_to_areas.get(dist):
    dist_to_areas[dist].append(area)
  else:
    dist_to_areas[dist] = [area]

sorted_dist_to_areas = sorted(dist_to_areas)
for dist_to_ares in sorted_dist_to_areas:
  dist_to_cloudlets = {}
  for area in dist_to_areas[dist_to_ares]:
    cloudlets_in_area = self.areas_collection.find_one({'_id':
str(area.latitude) + 'x' + str(area.longitude)}).get('cloudlets')
    if cloudlets_in_area:
      for cloudlet in cloudlets_in_area:
        dist_to_cloudlet = calculate_dist(request.latitude, request.lon-
gitude, cloudlet.get('latitude'), cloudlet.get('longitude'))
        if dist_to_cloudlets.get(dist_to_cloudlet):
          dist_to_cloudlets[dist_to_cloudlet].append(cloudlet.get('id'))
        else:
          dist_to_cloudlets[dist_to_cloudlet] = [cloudlet.get('id')]

  sorted_dist_to_cloudlets = sorted(dist_to_cloudlets)
  for dist in sorted_dist_to_cloudlets:
    left = searched_devices_count - len(nearest_cloudlets)
    for id in dist_to_cloudlets.get(dist)[:left]:
      nearest_cloudlets.append(cloudlets_service_pb2.Cloudlet(id=id))
    if len(nearest_cloudlets) >= searched_devices_count:
      break

  if len(nearest_cloudlets) >= searched_devices_count:
    break
```

Рисунок 3.7 – Алгоритм пошуку найближчих ПТ

Знайшовши дистанції від місця запиту до ВТ у цьому регіоні, Cloudlets Service аналогічно проводить сортування за їх зростанням, після цього знаходить ті, котрі є найближчими. Коли в регіоні відсутня необхідна чисельність ВТ, тоді пошук буде продовжено у інших регіонах до тих пір, поки не буде знайдено потрібну кількість.

3.4 Запуск туманної програми

Для того щоб ТОС Fog_Comp мала змогу здійснювати розгортання образів програм на ПТ, їй потрібен такий сервіс, котрий буде їх планувати та розгортати.

Для забезпечення цього в ТОС є Scheduling Service, котрий забезпечує метод Create-Cluster, який є віддаленим, для формування нового кластера IoT-обладнання. Даний метод гарантує виклик API Gateway, як тільки отримає потрібне повідомлення від хабу.

Найперше при викликанні методу для формування свіжого кластеру Scheduling Service здійснює виклик методу FindImagesService з метою виявлення образу додатку туману, котра треба для IoT Hub, щоб створити кластер. За умови, що образ було успішно виявлено, матиме місце виклик FindNearest, який є віддаленим методом, у Cloudlets Service, що вказує на ВТ, котрі містяться безпосередньо біля координат хабу. Якщо ВТ відшукані, Scheduling Service створює масив ID вузлів, а за потреби появи якихось помилок, чи коли образ додатку або вузли відсутні, надсилає помилку формування кластера (див. рис. 3.8).

```
images_service_response = self.images_service_stub.Find(scheduling_ser-
vice_pb2.Image(name=request.image))
if images_service_response.status.code != 200:
  status = scheduling_service_pb2.Response(code=images_service_re-
sponse.status.code, message=images_service_response.status.message)
  return scheduling_service_pb2.ResponseWithCluster(status=status)

cloudlets_service_response = self.cloudlets_service_stub.FindNear-
est(scheduling_service_pb2.Cloudlet(latitude=request.coordinates.lati-
tude,longitude=request.coordinates.longitude))
if cloudlets_service_response.status.code != 200:
  status = scheduling_service_pb2.Response(code=cloudlets_service_re-
sponse.status.code, message=cloudlets_service_response.status.message)
  return scheduling_service_pb2.ResponseWithCluster(status=status)

nearest_cloudlets_id = []
for cloudlet in cloudlets_service_response.cloudlets:
  nearest_cloudlets_id.append(cloudlet.id)
```

Рисунок 3.8 – Пошук образу програми та ПТ

Так як на одному ВТ може виконуватися багато варіантів різноманітних програм, і будь-який з них може потребувати відкритих портів обладнання для побудови зовнішньої взаємодії із програмою, Scheduling Service має сформувати перелік тих портів, котрі потрібні додатку і забезпечити доступ до цих портів. Він з цією метою користується даними щодо образу додатку, в котрій наявний опис комбінації контейнера, зокрема поле ExposedPorts, формує порти та протоколи,

котрі програма використовує. З тієї причини, що програми не раз застосовують ті ж самі порти, 24, 82 або 433 як варіант, відсутня можливість перенаправити трафік від порту ПТ до порту контейнера з тим же номером, оскільки в ОС заборонено використовувати один порт для декількох процесів. Для розв'язання цією проблеми, варто проводити перенаправлення трафіку із визначених вибірково портів ПТ, на ті порти, потрібні програмам. Для цього Scheduling Ser vice створює довідник endpoint spec, в котрому наявний перелік тих портів, котрих програма потребує, та застосовувані протоколи, не заповнюючи поле PublishedPort. Це каже менеджеру рою про потребу перенаправити трафік з непередбачено обраного порту. Код реалізації дій, що проводить Scheduling Service для ідентифікації потрібних портів програмі, наведено на рис.3.9.

```
endpoint_spec = {'Ports': []}
try:
  image = self.docker_client.images.get(self.registry_url + '/' + re-
quest.image)
  container_config = image.attrs.get('ContainerConfig')
  if container_config.get('ExposedPorts'):
    for key, _ in container_config.get('ExposedPorts').items():
      keys = key.split('/')
      endpoint_spec['Ports'].append({'PublishedPort': None, 'TargetPort':
int(keys[0]), 'Protocol': keys[1], 'PublishMode': 'host'})

except docker.errors.APIError as error:
  status = scheduling_service_pb2.Response(code=500, message=str(error))
  return scheduling_service_pb2.ResponseWithCluster(status=status)
except docker.errors.ImageNotFound:
  status = scheduling_service_pb2.Response(code=404, message='Image with
this name not found.')
  return scheduling_service_pb2.ResponseWithCluster(status=status)
```

Рисунок 3.9 – Визначення портів, що прослуховуються додатком

Після того, як Scheduling Service впевнився у тому, що потрібний образу додатку наявний, він виявив найближчі до IoT хабу ПТ і сформував перелік тих портів, котрі прослуховують додаток. Потім він починає розгортати додаток на якомусь із заданих ВТ (див. рис. 3.10).

```
try:
  cluster = self.docker_client.services.create(
    endpoint_spec=endpoint_spec,
    mode=docker.types.ServiceMode('replicated', replicas=0),
    image=self.registry_url + '/' + request.image)

  self.clusters_collection.update_one({'_id': cluster.attrs.get("ID")},
{'$set': {'cloudlets': nearest_cloudlets_id}}, upsert=True)

  for cloudlet_id in nearest_cloudlets_id:
    cloudlet_labels = self.docker_client.nodes.get(cloudlet_id).at-
trs.get('Spec').get('Labels')
    cloudlet_labels[cluster.attrs.get('ID')] = 'True'
    self.docker_client.nodes.get(cloudlet_id).update({'Availability':
'active', 'Labels': cloudlet_labels, 'Role': 'worker'})

  cluster.update(constraints=[f'node.labels.{cluster.at-
trs.get("ID")}==True'], mode=docker.types.ServiceMode('replicated', rep-
licas=1))

except Exception as error:
  cluster.remove()
  status = scheduling_service_pb2.Response(code=500, message='An internal
server error occurred.' + str(error))
  return scheduling_service_pb2.ResponseWithCluster(status=status)

cluster_task = cluster.tasks()[0]
state = cluster_task.get('Status').get('State')

status = scheduling_service_pb2.Response(code=201, message='Cluster of
IoT devices has been created successfully.')
return scheduling_service_pb2.ResponseWithCluster(status=status, clus-
ter=scheduling_service_pb2.Cluster(id=cluster.attrs.get("ID"),
state=state))
```

Рисунок 3.10 – Створення кластера IoT -пристроїв

З цією метою, передусім, формується служба Docker Swarm на основі відібраної інформації, але водночас число екземплярів програм буде рівне 0. Так вчинили, щоб виконання програм не проходило до тих пір, аж поки не відмітять особливими мітками необхідні ВТ. Після створення служби, її ID і ті ПТ, що перебувають у взаємозв’язку із ними, записуються для зберігання в БД з метою ймовірного майбутнього знищення кластера обладнання. Потім Scheduling Service із застосуванням ID ВТ відшукує ті ВТ, які є найближче і помічає їх міткою, що співпадає з ID служби. Мітки такого роду застосовуються менеджером рою, для формулювання ВТ, контейнери на яких дозволено виконувати. Після успішної установки міток Scheduling Service проводить оновлення службу, число екземплярів додатків стає рівне одиниці, у такий спосіб починаючи розгортання. При виникненні будь-яких помилок під час цього процесу проходить знищення сформованої служби і, у відповідь на повідомлення щодо формування кластера,

відповідна помилка буде повернута. За умови, що запуск додатку увінчався успіхом, Scheduling Service визначає стан функціонування служби і вертає його поряд з ID до API Gateway.

3.5 Розгортання системи

Процес розроблення якого-небудь програмного продукту складно уявити без різного роду тестування її складових частин. На цій стадії здійснюється перевірка якості написаного програмного коду і на скільки він відповідає вимогам, котрі визначені на етапі проектування системи. У подальших підрозділах буде здійснено розгортання ТОС Fog_Comp і загальне тестування її роботоздатності.

Для того щоб здійснити тестування ТОС, потрібно провести попередню підготовку і запустити на виконання її складові частини. Щоб спростити процес власне установки, налаштування і виконання ТОС Fog_Comp складено Docker образи елементів її ядра, котрі містяться у публічному сховищі Docker Hub. Кожен образ ґрунтується на python:3.7.7-slim-buster, котрий містить компілятор мови Python та множину компонентів, які потрібні для його якісного функціонування. На додачу до вбудованих складових, у контейнері будь-якого сервісу інсталюються пакети, що також необхідні для якісної роботи.

Щоб здійснити тестовий запуску елементів ядра ТОС Fog_Comp потрібно виконати таке.

1) Провести розгортання двох ВМ або двох серверів під ОС Ubuntu 20.04, із інстальованим та налагодженим SSH сервером. З метою зручності розуміння внесемо позначення буквами А та В серверів розгортання ТОС.

2) Одержати IP- адреси серверів, котрі вже розгорнуті. Для тестування з ВТ, які віддалені географічно, усе устаткування має володіти публічними IP адресами.

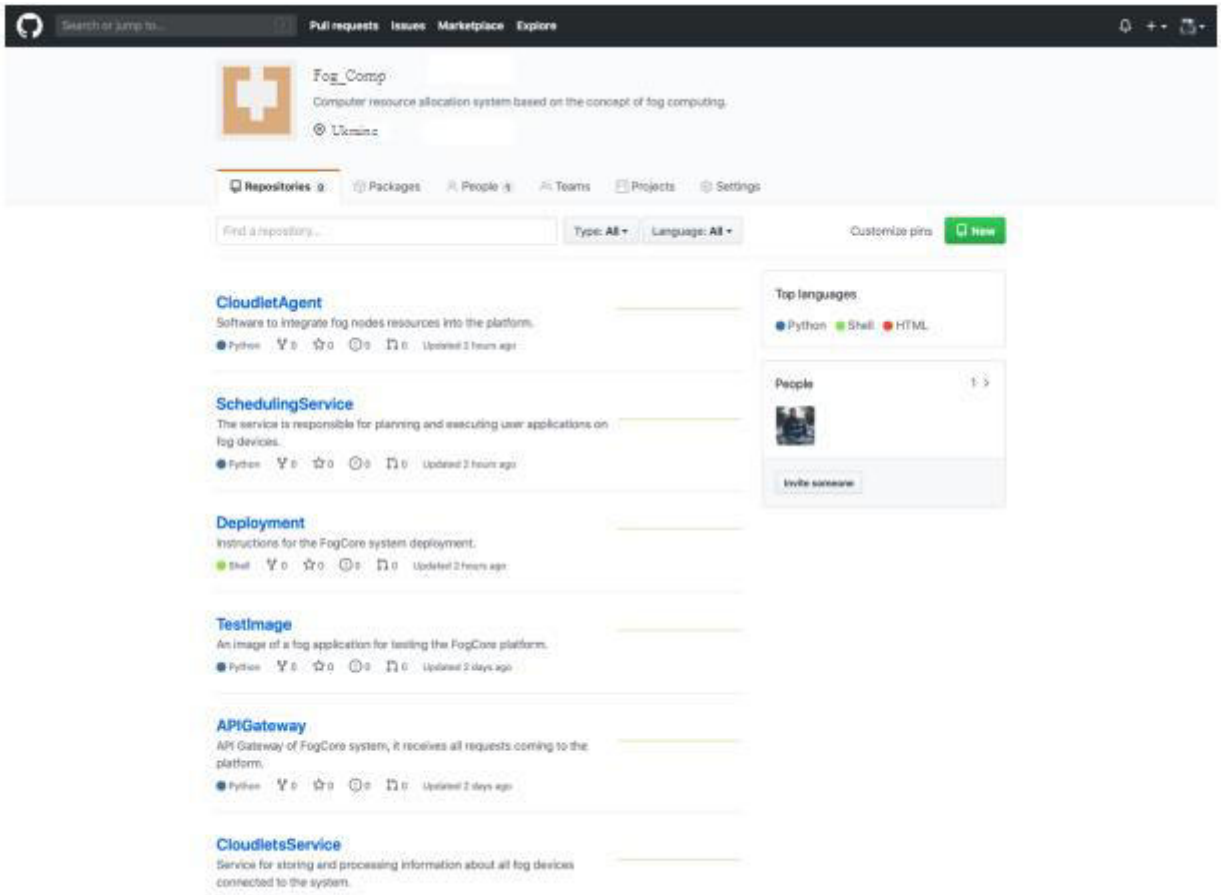

Рисунок 3.11 – Сторінка репозиторіїв ТОС Fog_Comp

Вихідний код всіх компонентів, включаючи інструкції англійською для розгортання ядра системи, а також встановлення та налаштування Cloudlet Agent на ПТ знаходиться у відкритому доступі на найбільшому хостингу ІТ проектів GitHub. Сторінку репозиторіїв компонентів системи Fog_Comp зображено на рис. 3.11.

3) Із сервером А здійснити під’єднаня за протоколом SSH.

4) Провести клонування сховища Deployment і перейти в папку, що задається при завантаженні файлів сховища, такою командою git clone https://github.com/Fog_Comp/Deployment.git && cd Deployment.

5) Інсталювати Docker Engine, виконавши сценарій docker_install.sk,./docker_install. sh.

6) Сформувати Docker томи, де будуть збережені дані елементів ТОС із використанням коду з рис. 3.12.

```
docker volume create --name=CloudletsServiceDB && docker volume create --
name=ImagesServiceDB && docker volume create --name=ImagesServiceRegistry
&& docker volume create --name=UsersServiceDB && docker volume create --
name=SchedulingServiceDB
```

Рисунок 3.12 – Команда для створення томів Docker контейнерів

Виконання багатьох контейнерів Docker може бути спрощеним із використанням засобу Docker Compose [32], котрий призначений для виявлення і запуску додатків для кількох контейнерів. Але для того, щоб його застосувати, потрібно перш за все заготувати налаштовувальні файли. Вони містяться в репозиторії, отже , вони були вже завантажені на сервер під час тго, як його клонували, проте окремі з них окремі потребують заміни.

7) У файлі dockerregistryconfig.yml зазначти IP адресу властиво сервера А значенням notifications -> endpoints -> url і auth -> token -> realm, полів проте значення портів не треба змінювати.

8) Аналогічно у файлі docker-compose.yml для поля services -> SchedulingService -> environment -> DOCKER_SWARM_MANAGER має бути встановлена IP адреса вже сервера В, а для поля services -> SchedulingService -> environment -> DOCKER REGISTRY URL - відповідно IP адреса сервера А.

9) Задати на виконання елементи ядра ТОС Fog_Comp кодом docker-compose up.

Інструмент, за необхідності, візьме образи елементів з сховища Docker Hub та запустить у автоматичному варіанті їх, застосовуючи для налагодження ті файли, котрі були заготовлені перед цим. Таким чином розгортання компонентів на сервері А завершено.

10) За протоколом SSH провести під’єднання до сервера В.

11) Провести клонування сховища Deployment і зайти в папку, що вказується при завантаженні файлів сховища, таким чином git clone https://github.com/Fog_Comp/Deployment.git && cd Deployment.

12) Виконати сценарій, наведений на рис. 3.13, для інсталювання та налаштування необхідних складових під root юзера

sudo./docker_swarm_manager_install. sh.

```
#!/bin/bash

echo
echo Docker Installation
DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" >/dev/null 2>&1 && pwd )"
$DIR/docker_install.sh

echo
echo Opening insecure access to the Docker Engine via HTTP
UNIT='docker.service'
DIR="/etc/systemd/system/${UNIT}.d"
sudo mkdir $DIR
{ echo "[Service]";
echo "ExecStart=";
echo "ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/contain-
erd/containerd.sock -H tcp://0.0.0.0:80";
} | sudo tee ${DIR}/override.conf
sudo systemctl daemon-reload

echo
echo Adding Insecure Docker Registry
read -p "Enter Docker Registry IP address or URL and port: " DOCKER_REG-
ISTRY_IP
{ echo "{ \"insecure-registries\":[\"$DOCKER_REGISTRY_IP\"] }"; } | sudo
tee /etc/docker/daemon.json

echo
echo Restart Docker
sudo service docker restart

echo
echo Enabling Swarm Mode
sudo docker swarm init --task-history-limit 0
```

Рисунок 3.13 – Скрипт встановлення та запуску Docker Swarm Manager

Даний сценарій перш за все проводить інсталяцію Docker Engine при допомозі сценарію docker install.sh. Далі змінюється файлу конфігурації Docker з метою відкриття доступу до Docker Engine за портом 80-м, це дасть змогу інакшим сервісам одержати до нього також доступ, застосовуюи RESTfull API. Для того, щоб ПТ і Swarm менеджер мали б змогу застосовувати з'єднання із Docker Registry, котре є незахищеним, та здобути доступ до образів програм туманів, сценарій хочу від адміністратора дозволу на ввід IP адресу та порту Registry, при чому адресою треба задати IP хоста A, тоді як значення порту має бути 4000. Скрипт виконує перезапуск Docker Engine, щоб внесені виправлення стали дійсними, потім здійснює його переведення в варіант функціонування Swarm як менеджера. Після цього приготовляння і запуск складових елементів ядра ТОС закінчено.

Щоб запустити ПТ треба реалізувати дії, відображені нижче.

1) Провести підготовку серверів або ВМ під управлінням ОС Ubuntu 20.04, забезпечити доступ до Інтернету, налаштувати SSH сервер, в чисельності потрібних ПТ.

2) Одержати серверні IP- адреси. Якщо ТОС тестується із вузлами, які є географічно далекі, потрібно публічні IP адреси використовувати.

3) Провести під'єднання по протоколу SSH до ВТ.

4) Здійснити клонування сховища Cloudlet Agent і перейти в папку, що задається при завантаженні файлів сховища, задавши такий код git clone https://github.com/Fog_Comp/CloudletAgent.git && cd CloudletAgent.

5) Від імені root юзера виконати код, наведений на рис. 3.14, для інсталяції і налагодження потрібних елементів sudo./install.sh.

```
#!/bin/bash

echo pip3 Installation
sudo apt install python3-pip -y

echo Python packages Installation
pip3 install grpcio protobuf docker

echo Docker Installation
DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" >/dev/null 2>&1 && pwd )"
$DIR/docker_install.sh

echo Adding insecure Docker Registry
read -p "Enter Docker Registry IP address or URL and port: " DOCKER_REGIS-
TRY_IP
{ echo "{ \"insecure-registries\":[\"$DOCKER_REGISTRY_IP\"] }"; } | sudo
tee /etc/docker/daemon.json

echo Docker Restart
sudo service docker restart
```

Рисунок 3.14 – Скрипт встановлення та запуску Cloudlet Agent

Цей сценарій здійснює встановлення менеджера пакетів, потім при допомозі тільки що проінстальованого менеджера pip3 інсталює python пакети, котрі потрібні для функціонування агента. Далі із використанням інакшого сценарію docker_install.sk інсталює Docker Engine, який потрібний для запуску завдань туману юзерів. Для того, щоб ВТ мали змогу співпрацювати, використовуючи

незахищений зв’язок з Docker Registry, треба, щоб адміністратор ввів IP адресу сервака А та порт 4000. Така інформація буде додана до файлу настройок Docker. Щоб ці зміни почали діяти, Docker Engine треба перезапустити.

Усі дії, котрі наведені вище, були втілені на двох хостах, що перевело ТОС Fog_Comp до такого статусу: усі елементи ядра запущені вдало на одній ВМ, окрім менеджера рою, котрий запущений у відокремленій ВМ на тому самому хості; система містить є 9 ВТ, які готові до реєстрації, всякий з них функціонує на окремому хості всередині своєї ВМ.

3.6 Додавання та запуск туманної програми

Щоб виконати туманні додатки потрібно доповнити ТОС бодай кількома ВТ, на котрих і будуть виконуватися образи програм. Ось чому у першому тесті буде досліджено потрібний процес реєстрування нового ВТ у ТОС, котрий проводиться адміністратором. Для реєстрації нових ПТ, адміністратор має під’єднатися до тих ПТ, які підготовлені за протоколом SSH на минулому кроці. Вподальшому при використанні стандартної команди Linux – export, встановити як величну оточення CLOUDLETS_SERVICE_URL IP адресу сервака А та через «:» задати порт 50050. Ця величина буде братися Cloudlet Agent для виклику способу реєстрації свіжого ПТ у Cloudlets Service.

Потім треба запустити агента, тому треба запустити на виконання файл app.py. Оскільки це перший такий запуск ПТ, Cloudlet Agent не знайде файл з даними про ПТ та реєстрацію. Адміністратор має йти за інструкціями на моніторі, перевірити та за потреби внести корективи щодо ВТ: його ім’я, число ядер та тактову частоту процесора, об’єм ОП та ПЗП, назву ОС та версію ядра, IP адресу, географічні координати, країну, місто та область, де міститься ВТ. Коли правильність даних підтверджена, ПТ буде зареєстровано в ТОС і надасть адміністратору стан дії. Якщо реєстрація пройшла благополучно, тоді

адміністратор має при допомозі бравзера перейти на сторінку ПТ і відшукати там серед різних пристроїв саме той, котрий щойно був ним успішно зареєстрований (див. рис. 3.15).

Щоб IoT -обладнанню вдалося почати процес розгортання додатків туману, при надсиланні відповідного запиту до API Gate way, програміст має помістити образ свого додатку. Для виконання процесу тестування ТОС Fog_Comp було сформовано Docker образ test image, котрий є звичайним Web-додатком для прослуховування 80 порта і який помістить на екран вітальний текст. Перш, ніж завантажити цей образ, розробник повинен пройти реєстрацію в ТОС.

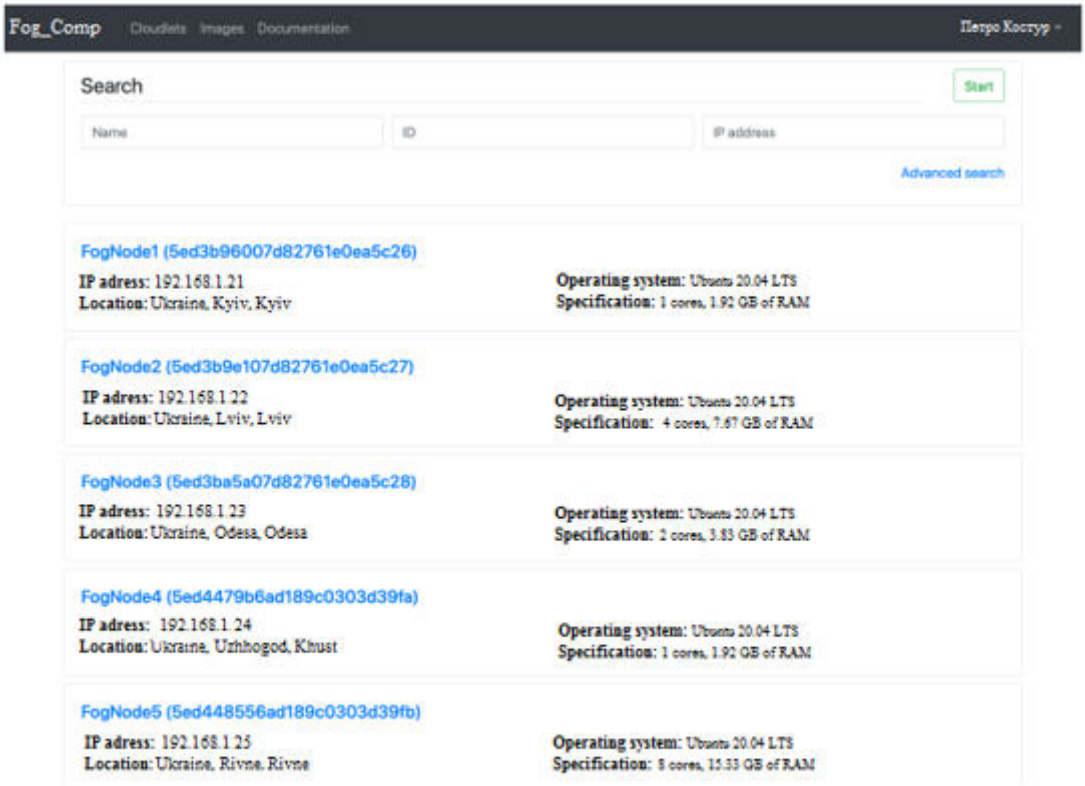

Рисунок 3.15 – Сторінка зареєстрованих ПТ

Щоб зареєструватися в ТОС користувачу потрібно із використанням веб-бравзера перейти на сторінку, показану на рис. 3.16. Тут необхідно внести дані до форми, задавши ім'я, прізвище, логін, пароль (не менше 8 символів). Якщо введений логін вже зайнятий чи буде неспівпадіння пароля та підтвердження вводу, на моніторі з’явиться текст про наявну помилку.

Якщо реєстрація пройшла успішно, розробник має бути автентифікованим із застосуванням Docker Client на своїй машині. Це робиться виконанням команди docker login з атрибутом IP адреси сервака A і порту 4000 так: registry_ip:port. Уподальшому фахівець має надати нове ім’я образу додатку командою docker tag таким чином reg-istry_ip:port/name:tag. Що повідомить Docker Client, у котре сховище потрібно помістити образ програми. Здійснити завантаження готового образу найпростіше при допомозі команди Docker Push

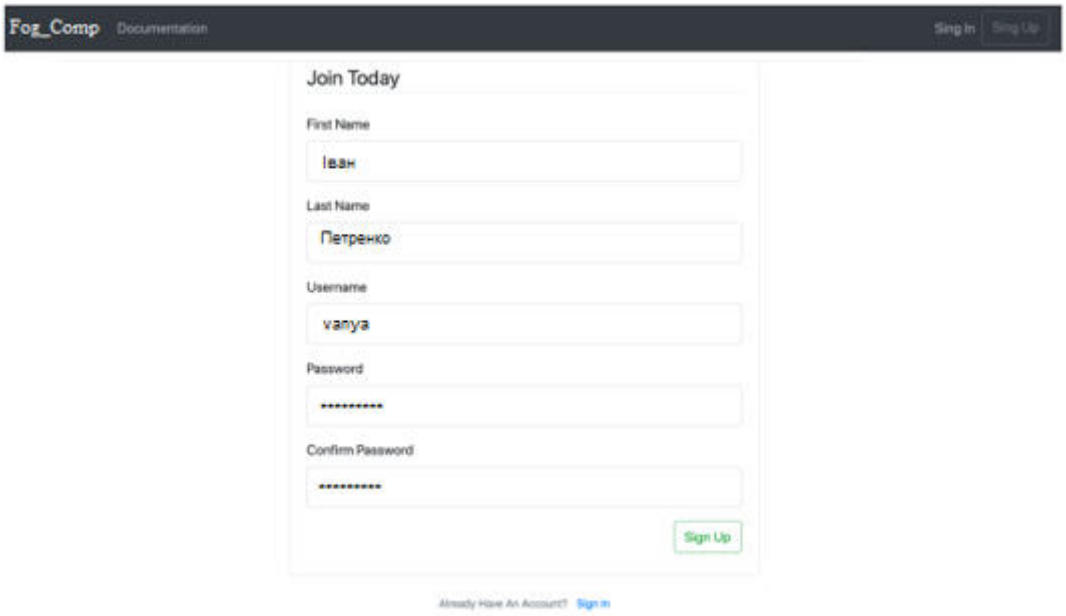

Рисунок 3.16 – Сторінка реєстрації в ТОС Fog_Comp

За умови, що завантаження успішно відбулося, програміст має побачити саме той образ, котрий ним тільки що був доданий, на сторінці образів програм туману (див. рис. 3.17).

Рисунок 3.17 – Сторінка образів туманних додатків

Відтак коли у системі є ПТ, котрі додав адміністратор, і образи програм, котрі завантажені програмістами, хаб може запитати доступ до потрібної програми шляхом надсилання відповідного HTTP- запиту до API Gateway. Для ознайомлення зі переліком методів API, котрі надає ТОС Fog_Comp, розробнику потрібно перейти на сторінку API власне розробника, наведену на рис. 3.18.

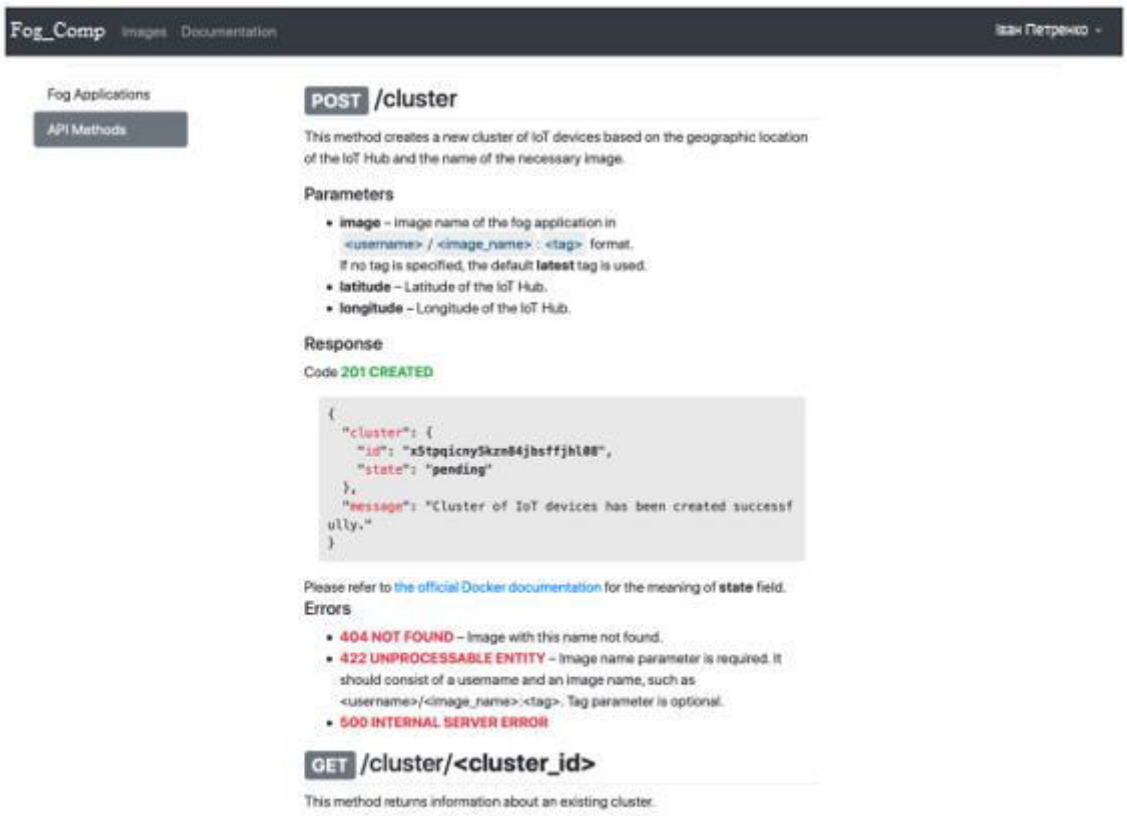

Рисунок 3.18 – Сторінка API розробника

Оскільки IoT -пристрої не входять до складових частин ТОС, тому для виконання тестування та симуляції запитів, котрі відправляє IoT Hub, застосовувалася консольна утиліта curl (рис. 3.19). Найперший запит є POST кодом для формування свіжого кластера IoT -обладнання. Атрибутом у цього запиту є ім’я того образу, котрий потребує розгортання на ПТ, та ширина і довгота того місця, в котрому необхідна програма туману. У полі на рис. 3.19 назва образу – testimage, а широта і довгота – це координати 1-го корпусу університету. У відповідь на цей запит ТОС повернула HTTP повідомлення із JSON форматом, котре містить ID з кластера та його статус. Наступний запит є GET запитом, котрий призначений для одержання інформації щодо кластера. Для отримання цих даних потрібно передати ID кластера як фрагмент URL стрічки. Відповіддю ТОС на цей запит буде повідомлення JSON формату, котре має дані щодо IP адреси ВТ, на котрому програма була розгорнута, її ім'я, статус її функціонування та набір відкритих портів для проведення взаємодії складових. IP адреса ПТ показує, що ТОС було розгорнуто програму, котру запитують, на ВТ, який є найближчому до точки вимоги.

```
MacBook-Pro:~$ curl -X "POST" "http://192.168.1.3:81/cluster?image
{
  "cluster": {
    "id": "l4c2t1aa11v4ytbsv7ok3alsw",
    "state": "pending"
  },
  "message": "Cluster of IoT devices has been created successfully."
}
MacBook-Pro:~$ curl "http://192.168.1.3:81/cluster/l4c2t1aa11v4ytbsv7ok3alsw"
{
  "cluster": {
    "cloudlet_ip": "192.168.1.22",
    "exposed_ports": [
      {
        "protocol": "tcp",
        "published_port": 32769,
        "target_port": 80
      },
      {
        "protocol": "tcp",
        "published_port": 32768,
        "target_port": 443
      }
    ],
    "id": "l4c2t1aa11v4ytbsv7ok3alsw",
    "image": "test_image",
    "state": "running"
  },
  "message": "IoT devices cluster was found."
}
```

Рисунок 3.19 – Створення кластера IoT –пристроїв

3.7 Висновок до третього розділу

У цьому розділі були описані складові частини програмного виконання компонентів розроблюваної ТОС Fog_Comp, котрі потрібні для втілення варіантів використання, запропонованих раніше. Також наведено способи використання інструментів та механізмів підтримки функціонування ТО, вибраних для проведення дослідження.

Ядро ТОС Fog_Comp успішно було запущено на двох ВМ після здійснення описаних вище дій. ТОС містить 9 зареєстрованих ПТ, образи програм, завантажені розробником, та кластер IoT-обладнання, котрий було розгорнуто на ВТ, що є самим ближчим до місця вимоги ВТ. Це значить, що тестування розробки є успішним та всі складові ТОС функціонують згідно зі специфікацією, котра була сформована на етапі проектування.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

4.1 Охорона праці

Метою кваліфікаційної роботи магістра є розробка ТОС, що пропонує ОР на базі концепції ТО, яка повинна контролювати, налаштовувати та надавати туманні ОР в автоматичному режимі. Оскільки, проведення робіт з розробки та використання ТОС передбачає застосування комп’ютерної техніки, зокрема ПК та периферійних пристроїв, то обов’язковим є дотримання вимог з охорони праці і техніки безпеки.

Для ефективної і безпечної роботи колективу працівників з розробки ПЗ комп’ютерних систем, в тому числі і фахівців з підвищення ефективності контролю доступу в приміщення, необхідно організувати безпечні умови праці. При цьому керівник організації несе безпосередню відповідальність за порушення нормативно-правових актів з охорони праці [33]. Окрім цього, на робочих місцях працівників необхідно забезпечити дотримання вимог, затверджених Наказом Мінсоцполітики від 14.02.2018 за № 207 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями». Згідно Вимог приміщення, де розміщені робочі місця операторів, крім приміщень, у яких розміщені робочі місця операторів великих ЕОМ загального призначення (сервер), мають бути оснащені системою автоматичної пожежної сигналізації відповідно до цих вимог;

– переліку однотипних за призначенням об'єктів, які підлягають обладнанню автоматичними установками пожежогасіння та пожежної сигналізації, затвердженого наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 22.08.2005 N 161, зареєстрованого в Міністерстві юстиції України 05.09.2005 за N 990/11270 (НАПБ Б.06.004-2005);

– Державних будівельних норм "Інженерне обладнання будинків і споруд. Пожежна автоматика будинків і споруд", затверджених наказом Держбуду України

від 28.10.98 N 247 (далі - ДБН В.2.5-56:2014, з димовими пожежними сповіщувачами та переносними вуглекислотними вогнегасниками.

В інших приміщеннях допускається встановлювати теплові пожежні сповіщувачі. Приміщення, де розміщені робочі місця операторів, мають бути оснащені вогнегасниками, кількість яких визначається згідно з вимогами ДСТУ 4297:2004 «Пожежна техніка. Технічне обслуговування вогнегасників». Загальні технічні вимоги і з урахуванням граничнодопустимих концентрацій вогнегасної рідини відповідно до вимог НАПБ А.01.001-2014. Приміщення, в яких розміщуються робочі місця операторів сервера загального призначення, обладнуються системою автоматичної пожежної сигналізації та засобами пожежогасіння відповідно до вимог ДБН В.2.5-56:2014, ДБН В.2.5-56:2010, НАПБ А.01.001-2014 і вимог нормативно-технічної та експлуатаційної документації виробника. Проходи до засобів пожежогасіння мають бути вільними.

Лінія електромережі для живлення комп’ютера та периферійних пристроїв повинні бути виконаними як окрема групова трипровідна мережа шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Не допускається використовувати нульовий робочий провідник як нульовий захисний провідник. Нульовий захисний провідник прокладається від стійки групового розподільного щита, розподільного пункту до розеток електроживлення. Не допускається підключати на щиті до одного контактного затискача нульовий робочий та нульовий захисний провідники.

Площа перерізу нульового робочого та нульового захисного провідника в груповій трипровідній мережі має бути не менше площі перерізу фазового провідника. Усі провідники мають відповідати номінальним параметрам мережі та навантаження, умовам навколишнього середовища, умовам розподілу провідників, температурному режиму та типам апаратури захисту, вимогам НПАОП 40.1-1.01-97.

У приміщенні, де одночасно експлуатуються понад п’ять комп’ютерів, на помітному, доступному місці встановлюється аварійний резервний вимикач, який

може повністю вимкнути електричне живлення приміщення, крім освітлення. Комп’ютери повинні підключатися до електромережі тільки за допомогою справних штепсельних з'єднань і електророзеток заводського виготовлення.

У штепсельних з'єднаннях та електророзетках, крім контактів фазового та нульового робочого провідників, мають бути спеціальні контакти для підключення нульового захисного провідника. Їхня конструкція має бути такою, щоб приєднання нульового захисного провідника відбувалося раніше, ніж приєднання фазового та нульового робочого провідників. Порядок роз’єднання при відключенні має бути зворотним. Не допускається підключати комп’ютери до звичайної двопровідної електромережі, в тому числі – з використанням перехідних пристроїв. Електромережі штепсельних з'єднань та електророзеток для живлення комп’ютерної техніки повинні бути виконаними за магістральною схемою, по 3-6 з'єднань або електророзеток в одному колі. Штепсельні з'єднання та електророзетки для напруги 12 В та 42 В за своєю конструкцією мають відрізнятися від штепсельних з'єднань для напруги 127 В та 220 В. Штепсельні з'єднання та електророзетки, розраховані на напругу 12 В та 42 В, мають візуально (за кольором) відрізнятися від кольору штепсельних з'єднань, розрахованих на напругу 127 В та 220 В.

При підвищенні ефективності контролю доступу в приміщення, де для забезпечення безпеки мешканців, співробітників і збереження майна використовуються ДС, важливим, з точки зору охорони праці, є забезпечення достатньої величини природного та штучного освітлення, які визначені у НПАОП 0.00-7.15-18. Організація робочого місця фахівця із дослідження методів та програмно-апаратних засобів оптимізаційних процесів на основі ГА повинна забезпечувати відповідність усіх елементів робочого місця та їх розташування ергономічним вимогам ДСТУ 8604:2015 «Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги». Відстань від екрана до ока фахівців, які працюють за комп’ютером визначається згідно з вимогами ДСанПіН 3.3.2.007-98.

Розміщення принтера або іншого пристрою введення-виведення інформації

на робочому місці має забезпечувати добру видимість екрана комп'ютера, зручність ручного керування пристроєм введення-виведення інформації в зоні досяжності моторного поля згідно з вимогами ДСанПіН 3.3.2.007-98.

Таким чином, у результаті аналізу вимог щодо охорони праці користувачів комп'ютерів, визначено особливості організації робочих місць, вимог з електробезпеки, природного та штучного освітлення для ефективної і безпечної роботи фахівців з розробка ТОС, що пропонує ОР на базі концепції ТО, яка повинна контролювати, налаштовувати та надавати туманні ОР в автоматичному режимі.

4.2. Комп'ютерне забезпечення процесу оцінки радіаційної та хімічної обстановки

Екологічне співтовариство розробило сімейство інструментів комплексної екологічної оцінки. Програмне забезпечення і послуги (ESS), комерційна група IIASA, включаючи AirWare (для повітряних проблеми якості), WaterWare (для якості води), CityWare (якість повітря і води в контексті великих міст) і EIAxpert (для надання допомоги із загальним впливом на навколишнє середовище). Функціональність в цілому схожа на RAISON, хоча з великим акцентом на моделювання і меншим акцентом на керування даними. Знову ж таки, інструменти ESS розроблені як модульні набори інструментів (доступні спеціальні системи для вирішення конкретних завдань). Компоненти включають стандартні імітаційні моделі, включаючи моделі ISC і PBM Агентства з охорони навколишнього середовища США, управління даними, в тому числі ГІС, аналіз даних (наприклад, аналіз часових рядів даних спостережень), візуалізація, а також оптимізація [34].

Іноді немає готових моделей, придатних для конкретного застосування, але тягар розробки нової програми на Фортрані або C / C ++ є надмірним. Розробка моделі оточення може відносно легко реалізувати власні моделі комп'ютерів і не турбуватися про включення процедур для вирішення рівнянь, візуалізації і т. д. Як

правило, за допомогою цих інструментів користувач просто повинен вказати свою модель, використовуючи або математичні рівняння, або спеціальні графічні символи або значки, які безпосередньо представляють поведінку системи.

На даний момент є розроблені моделі комп'ютерного забезпечення процесу для оцінки радіаційної та хімічної обстановки.

GEMS – це система на основі моделей, яка підтримує оцінки схильності і ризику, надаючи доступ до одиночних і мультимедійних моделям експозиції, фізико-хімічні властивості методи оцінки, статистичний аналіз, графічні та картографічні програми з відповідними даними на навколишнє середовище, джерела, рецептори і популяції. У розробці з 1981 року, GEMS надає аналітикам 84 84 інтерактивний, легко досліджуваний інтерфейс для різних моделей, програм і даних, які необхідні для оцінки хімічного впливу і ризику [34].

HSPF – це комплексний пакет для моделювання кількості і якості стоків з багатоцільових водозборів і процесів радіації, що відбуваються в потоках або повністю змішаних озерах. Це дозволяє інтегроване моделювання землі і грунту, процесів забруднення при гідравлічній і осадово-хімічній взаємодії. Результатом моделювання є тимчасові дані витрати стоку, концентрація поживних речовин і пестицидів, а також дані кількості і якості води в будь-якій точці водозбору. Алгоритми якості води включають динаміку BOD / DO, вуглець, азот і фосфор. Процеси трансформації, які включені в модель це: гідроліз, фотоліз, окислення, випаровування, сорбція і биодеградація. Вторинні або «дочірні» хімічні речовини також моделюються.

Вимоги до даних для моделі можуть бути досить широкими в залежності від конкретного застосування.

Модель MMSOILS – це методологія оцінки впливу на людину і ризику для здоров'я, пов'язаних з викидами забруднень з небезпечних відходів. Мультимедійна модель, що стосується перенесення хімічної речовини в ґрунтові води, поверхневі води, атмосферу і накопичення в їжі. Шляхи впливу на людину, які розглянуті в методології включають: потрапляння в грунт, вдихання летких речовин в повітря і тверді частинки, шкірний контакт, прийом питної води і т.д.

Ризик, пов'язаний із загальною дозою опромінення, розраховується на основі хімічної токсичності [34].

4.3 Висновок до четвертого розділу

В цьому розділі проаналізовано важливі питання охорони праці та безпеки в надзвичайних ситуаціях, висвітлено питання комп’ютерного забезпечення процесу оцінки радіаційної та хімічної обстановки.

ВИСНОВОК

У результаті виконаної роботи було розроблено ТОС Fog_Comp, котра пропонує туманні ОР в автоматичному режимі із використанням концепції ТО.

Основні результати проведеного дослідження:

– проаналізовано предметну область, проведено огляд літературних джерел щодо ТО, розглянуто основні технології втілення ТО та забезпечення їх комунікації, досліджено існуюче ПЗ для коректної роботи з ТО;

– сформовано вимоги до ТОС Fog_Comp (функціональні та нефункціональні), здійснено її проектування;

– запропоновані основні актори та сформовані їхні можливості;

– спроектовано усі складові частини системи, приведено основні варіанти використання розробки;

– розглянуті програмні компоненти ТОС Fog_Comp;

– ядро розробленого програмного продукту було успішно запущено на двох серверах;

– Fog_Comp складається із дев’яти зареєстрованих ПТ, образів додатків, котрі завантажені самим розробником, та кластера IoT-обладнання, який також був розгорнутий успішно на найближчих до місця вимоги ВТ серверах.

Це свідчить про успішність тестування ТОС Fog_Comp та можливість її подальшого застосування.

Разом з тим є можливі шляхи покращення ТОС у частині:

– забезпечення збереження даних програм туману на сусідніх ПТ із забезпеченням їх повного відновлення якщо ВТ вийде з ладу;

– втілення вибирання найближчих ПТ, котрі відповідають визначеним параметрам;

– здійснення взаємодії із кластерами IoT-обладнання при допомозі ТД.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Industrial Internet Consortium. URL: https://www.iiconsortium.org/index.htm (дата звернення: 01.12.2023).

2. Iorga M. Fog computing conceptual model. Gaithersburg, MD, 2018.

3. Shi W. Edge Computing: Vision and Challenges // IEEE Internet of Things Journal. 2016. Vol. 3, No 5. P. 637–646.

4. OpenFog Consortium Architecture Working Group OpenFog Reference Architecture for Fog Computing // OpenFog. 2017. P. 162.

5. Bonomi F. Fog Computing: A Platform for Internet of Things and Analytics 2014. 169–186 p.

6. Bonomi F. Fog computing and its role in the internet of things // Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC ’12, 2012, New York, New York, USA. ACM Press, 2012. P. 13.

7. Yu T., Wang X., Shami A. A Novel Fog Computing Enabled Temporal Data Reduction Scheme in IoT Systems // GLOBECOM 2017 - 2017 IEEE Global Communications Conference, 2017. IEEE, 2017. P. 1–5.

8. Hu P. Survey on fog computing: architecture, key technologies, applications and open issues // Journal of Network and Computer Applications. 2017. Vol. 98. P. 27–42.

9. Rabay’a A., Schleicher E., Graffi K. Fog Computing with P2P: Enhancing Fog Computing Bandwidth for IoT Scenarios // 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2019. IEEE, P. 82–89.

10. Alam M. G. R., Tun Y. K., Hong C. S. Multi-agent and reinforcement learning based code offloading in mobile fog // 2016 International Conference on Information Networking (ICOIN), 2016. IEEE, 2016. P. 285–290.

11. Костур П.М. Технології підтримки туманних обчислень // Інформаційні моделі, системи та технології: Праці XI наук.-техн. конф. Тернопіль, 2023. С. 204.

12. Empowering App Development for Developers | Docker. URL: https://www.docker.com/ (дата звернення: 03.12.2023).

13. Swarm mode overview | Docker Documentation. URL: https://docs.docker.com/engine/swarm/ (дата звернення: 07.12.2023).

14. Production-Grade Container Orchestration - Kubernetes. URL: https://kubernetes.io/ (дата звернення: 07.12.2023).

15. Docker Registry | Docker Documentation. URL: https://docs.docker.com/registry/ (дата звернення: 08.12.2023).

16. Fielding R. T., Taylor R. N. Principled design of the modern Web architecture // ACM Transactions on Internet Technology (TOIT). 2002. Vol. 2, No 2. P. 115–150.

17. Tay B. H., Ananda A. L. A survey of remote procedure calls // ACM SIGOPS Operating Systems Review. 1990. Vol. 24, No 3. P. 68–79.

18. gRPC. URL: https://grpc.io/ (дата звернення: 12.12.2023).

19. Protocol Buffers | Google Developers. URL: https://developers.google.com/protocol-buffers (дата звернення: 12.12.2023).

20. Ionescu V. M. The analysis of the performance of RabbitMQ and ActiveMQ // 2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER), 2015. IEEE, 2015. P. 132–137.

21. Messaging that just works — RabbitMQ. URL: https://www.rabbitmq.com/ (дата звернення: 13.12.2023).

22. Karakaya Z., Yazici A., Alayyoub M. A Comparison of Stream Processing Frameworks // 2017 International Conference on Computer and Applications (ICCA), 2017. IEEE, 2017. P. 1–12.

23. Apache Kafka. URL: https://kafka.apache.org/ (дата звернення: 13.12.2023).

24. Eclipse ioFog. URL: https://iofog.org/ (дата звернення: 13.12.2023).

25. Sonm — Decentralized Fog Computing Platform. URL: https://sonm.com/ (дата звернення: 14.12.2023).

26. Difuon - Blockchain Fog / Edge Computing Platform and Marketplace. URL: https://difuon.com/ (дата звернення: 14.12.2023).

27. AWS IoT Greengrass – Amazon Web Services. URL:

https://aws.amazon.com/greengrass/ (дата звернення: 14.12.2023).

28. Azure IoT Edge | Microsoft Azure. URL: https://azure.microsoft.com/ services/ iot-edge/ (дата звернення: 14.12.2023).

29. Zhao J. T., Jing S. Y., Jiang L. Z. Management of API Gateway Based on Micro-service Architecture // Journal of Physics: Conference Series. 2018. Vol. 1087. P. 032032.

30. Welcome to Python.org. URL: https://www.python.org/ (дата звернення: 16.12.2023).

31. The most popular database for modern apps | MongoDB. URL: https://www.mongodb.com/ (дата звернення: 16.12.2023).

32. Overview of Docker Compose | Docker Documentation. URL: https://docs.docker.com/compose/ (дата звернення: 16.12.2023).

33. Толок А.О. Крюковська О.А. Безпека життєдіяльності: Навч. посібник. 2011. – 215 с.

34. Зеркалов Д.В. Охорона праці в галузі: Загальні вимоги. Навчальний посібник. – К.: Основа. 2011. – 551 с.

ДОДАТОК А

Тези конференції

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XI НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

13-14 грудня 2023 року

ТЕРНОПІЛЬ
2023

УДК 378.147
П.М. Костур
Тернопільський національний технічний університет імені Івана Пулюя, Україна

ТЕХНОЛОГІЇ ПІДТРИМКИ ТУМАННИХ ОБЧИСЛЕНЬ

P.M. Kostur

FOG COMPUTING SUPPORT TECHNOLOGIES

Організація туманних обчислень є нетривіальним завданням, особливо якщо вона має відбуватися автоматично. Якнайменше для її втілення потрібно відшукати такі вузли туману, які є найближчими до місця виникнення обчислювальних ресурсів, встановити на них необхідні додатки, забезпечити стеження за ними та підтримку їх коректного функціонування. Для забезпечення всього, наведеного вище, повинна бути складна система, котра містить множину компонентів, котрі перебувають у взаємодії один з одним. Зараз наявна значна кількість різноманітних програмних інструментів, при використанні яких обчислювальна система буде так чи інакше забезпечувати виконання тих завдань, які встановлені перед нею.

Для роботи різноманітних додатків на вузлі туману можна застосовувати відкрите програмне забезпечення (ПЗ) для розроблення, доставляння та завантаження програм – Docker. Він дає змогу пакувати та виконувати програми у контейнерах, котрі дозволяють відділити програми власне від інфраструктури, яка їх оточує.

Застосування критеріїв ізоляції та безпеки дають змогу виконувати разом на одному хості декілька контейнерів. Контейнери легковагові, оскільки вони не потребують гіпервізора, виконуються безпосередньо на ядрі.

Виконання контейнерів та їх підтримування у такому вигляді не можна проводити вручну, оскільки їх чисельність стає аномальною. Docker Swarm якраз і призначений, щоб автоматизувати такі процеси. Це ПЗ надає множину засобів для масштабування, формування мереж, захисту та підтримування додатків у контейнерах, на додачу до властивостей контейнерів як таких. Цей засіб вмонтований у Docker Engine по дефолту і є досить простим у розумінні.

Для тонкого налаштування та керування контейнерами застосовують комплексні інструменти, з них найбільш поширеним є Kubernetes. Застосовується для оркестрації контейнерів для забезпечення автоматизованого процесу розгортання, масштабування та керування застосунками на базі контейнерів. Для функціонування Kubernetes потрібно задати стан кластера, а саме вказати програми, їх чисельність для запуску, образи яких контейнерів застосовувати тощо. Після цього інструмент забезпечуватиме оптимальну підтримку кластера у визначеному стані. Платформа управляє не окремими контейнерами, а їх зв'язаними наборами, котрі носять назву Pod. Вони є величиною планування в Kubernetes. В цьому полягає відмінність від Docker Swarm.

Для розробленої туманної системи обчислень функціональності Docker Swarm, буде достатньо, тому як основний засіб для оркестрації обраний саме він. З метою доставляння та виконання різних додатків на необхідних ВТ, Docker Swarm потребує сховища, у котрому міститимуться всі потрібні образи додатків. Тут зручно використати Docker Registry, котрий є масштабованим open-source серверним застосунком для зберігання і розповсюдження Docker- образів. На його базі функціонує Docker Hub, який є найбільшим у світі репозиторієм контейнерних образів.

СЕКЦІЯ 4. ПРОГРАМНА ІНЖЕНЕРІЯ ТА МОДЕЛЮВАННЯ СКЛАДНИХ РОЗПОДІЛЕНИХ СИСТЕМ