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

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту
допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
"___"_____2019р.

Магістерська дисертація

зі спеціальності - 122 Комп'ютерні науки
за спеціалізацією - Комп'ютерний моніторинг та геометричне моделювання процесів і систем

на тему:
"Мікросервіс обробки геоінформаційних даних в середовищі хмарних обчислень"

Виконав : студент ____6__курсу, групи__ТР-81МП____

_____Будько Дмитро Ігорович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник
_____доцент, к.т.н Смаковський Д. С._____ _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент_____ _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
„Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти другий ріень, магістерський

зі спеціальності - 122 Комп’ютерні науки
за спеціалізацією - Комп’ютерний моніторинг та геометричне моделювання процесів і систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ

Будьку Дмитру Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема дисертації Мікросервіс обробки геоінформаційних даних в середовищі хмарних обчислень
Науковий керівник Смаковський Денис Сергійович к.т.н., доцент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 20__р. № _____
2. Строк подання студентом дисертації _____
3. Об’єкт дослідження: мікросервісна архітектура в розробці програмного забезпечення
4. Предмет дослідження: використання мікросервісної архітектури для вирішення задач геоінформаційної інженерії
5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): проектування та розробка мікросервісу для обробки геоінформаційних даних, розміщення мікросервісу в хмарному середовищі OpenStack з використанням фреймворку kubernetes.
6. Перелік ілюстраційного матеріалу цілі та задачі, розподілені системи, середовище розробки, use case діаграми, приклади роботи програмного модулю.

7. Перелік публікацій
Тези: “Мікросервіс обробки геоінформаційних даних у середовищі хмарних обчислень”. Тези доповіді XVII міжнародної науково-практичної конференції__
8. Дата видачі завдання ”_20_”__вересня__2018 р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	15.10.2018	
2.	Вивчення та аналіз задачі	20.12.2018	
3.	Розробка архітектури та загальної структури системи	14.03.2019	
4.	Розробка структур окремих підсистем	13.06.2019	
5.	Програмна реалізація системи	17.09.2019	
6.	Оформлення пояснювальної записки	21.10.2019	
7.	Захист програмного продукту	07.11.2019	
8.	Передзахист	20.11.2019	
9.	Захист		

Студент _____ _____Будько Д.І._____
(підпис) (прізвище та ініціали,)
Керівник роботи _____ _____Смаковський Д.С._____
(підпис) (прізвище та ініціали,)

РЕФЕРАТ

Структура і обсяг роботи.

Магістерська дисертація складається зі вступу, 6 розділів, висновку, переліку посилань з 13 найменувань, 2 додатки, і містить 34 рисунків, 23 таблиці. Повний обсяг магістерської дисертації складає 109 сторінок, з яких перелік посилань займає 2 сторінки, додатки – 23 сторінки.

Актуальність

В задачах геоінформаційної інженерії все більше з’являється задач, які потребують значної обчислювальної потужності, та потребують правильного структурування системи у зв’язку зі складністю проектування системи. Мікросервісна архітектура є одним із найкращих підходів для вирішення наведених проблем, тому була вибрана для дослідження.

Мета і завдання

Метою розробки є реалізація мікросервісів обробки геоінформаційних даних, який буде використовуватись для збору та обробки даних з геоінформаційних пристроїв.

Для досягнення поставленої мети були сформульовані наступні завдання дослідження, що визначили структуру дослідження:

— проаналізувати вже існуючі програмні рішення;

— проаналізувати літературні джерела на відповідну тематику;

— обрати засоби та інструменти розробки програмного продукту;

— обрати методи розробки;

— здійснити програмну реалізацію системи керування проектами кафедри з використанням обраних технологій програмування;

Вирішення поставлених завдань і досягнуті результати

Результатом є програмна система для збору і обробки геоінформаційних даних, реалізована за допомогою мікросервісної архітектури.

Об’єкт досліджень

Об’єктом дослідження є мікросервісна архітектура в розробці програмного забезпечення.

Предмет досліджень

Предметом дослідження є використання мікросервісної архітектури для вирішення задач геоінформаційної інженерії.

Методи дослідження

Розв’язання поставлених задач виконувались з використанням наступного методу:

- використання мікросервісної архітектури.

Практичне значення одержаних результатів роботи полягає в розробці системи обробки геоінформаційних даних, надання можливості збору та збереження вимірів, отриманих за допомогою спеціальних пристроїв. Надання зручного інтерфейсу для керування та обробки зібраних даних. Керування користувачами та пристроями.

Ключові слова.
МІКРОСЕРВІС, ХМАРНЕ СЕРЕДОВИЩЕ, KUBERNETES, ГЕОІНФОРМАЦІЙНІ ДАНІ.

ABSTRACT

Structure and scope of work.

The master's thesis consists of an introduction, 6 sections, conclusion, a list of references of 13 titles, 2 appendices, and contains 34 figures, 23 tables. The full volume of the master's thesis is 109 pages, of which the list of links occupies 2 pages, the annexes - 23 pages.

Topicality.

Geo-information engineering tasks are increasingly tasked with significant computational power and need to properly structure the system due to the complexity of system design. Microservice architecture is one of the best approaches to solve these problems, so it was selected for research.

Purpose and tasks.

The purpose of the development is to implement a microservice of geoinformation data processing that will be used to collect and process data from geoinformation devices.
The purpose of the development is to implement a microservice of geoinformation data processing that will be used to collect and process data from geoinformation devices.

To achieve this goal, the following research objectives were formulated, which determined the structure of the study:

- analyze already existing software solutions;
- to analyze literature sources on relevant topics;
- choose tools and tools for software development;
- choose methods of development;
- to implement software implementation of the department's project management system using selected programming technologies;

Problem solving and results achieved.

The result is a software system for the collection and processing of geoinformation data, implemented using microservice architecture.

Object of research.

The object of research is microservice architecture in software development.

The subject of research.

The subject of research is the use of microservice architecture to solve the problems of geoinformation engineering.

Research methods.

The tasks were solved using the following method:

- use of microservice architecture.

The practical significance of the results obtained is to develop systems for processing geoinformation data, providing the ability to collect and save measurements obtained through special devices. Provide a user-friendly interface for managing and processing the collected data. Manage users and devices.

Keywords.

MICROSERVICE, CLOUD ENVIRONMENT, KUBERNETES, GEOINFORMATION DATA.

ЗМІСТ

ВСТУП

Прогрес розвитку сучасних засобів реалізації програмних продуктів вносить багато інноваційних та прогресивних технологій і методологій, які можуть бути використані у великих, сучасних програмних проектах. Одними із основних переваг, які можуть бути корисними при розробці сучасних технологічних проектів – є покращення якості, збільшення швидкості розробки, зменшення кількості помилок, надання розробникам гнучкості при вирішенні основних бізнес задач.

Однією із найбільш прогресивних сучасних методологій розробки є так звана мікросервісна архітектура. На даний момент мікросервісна архітектура знаходиться в центрі уваги багатьох експертів з інформаційних технологій – цей підхід до розробки часто згадується в різноманітних блогах, публікаціях, статтях, дискусіях та на конференціях.

Поняття «Мікросервіс» або «архітектура мікросервісів» вперше з’явилося близько 2010го року, як опис особливого архітектурного стилю, який використовувався для розробки додатків. Такий підхід набув популярності через розвиток різноманітних практик для так званої гнучкої розробки. Такий підхід має ряд переваг, якщо мова йде про досягнення гнучкої розробки і повної безперервної доставки програмного продукту.

Можна сказати коротко, що мікросервісний стиль в розробці програмного забезпечення – це підхід, в якому додаток складається як певна сукупність малих, незалежних, не дуже зв’язаних сервісів між собою, які спілкуються одне з одним з використанням таких механізмів як, наприклад, Kafka, RabbitMQ, HTTP тощо. Важливою особливістю є незалежне розготання сервіса та використання майже повністю автоматичного середовища.

Однією з проблем, які можуть бути вирішені за допомогою вказаного вище

підходу є задача розробки програмного додатку для збору, збереження, обробки геоіформаційних даних: вимірів, пристроїв, геопозіціонування, користувацьких даних.

Пояснювальна записка містить вступ, п'ять розділів, висновки, список використаних джерел та п'ять додатків. Перший розділ містить постановку задачі розроблюваної системи. Другий розділ описує програмні рішення, що вже існуть. В третьому розділі описуються методи та інструменти які використовувались для розробки автоматизованої системи. Четвертий розділ містить опис програмної реалізації системи. П’ятий розділ описує методику роботи з розробленим застосунком.

1. ЗАДАЧА РОЗРОБКИ МІКРОСЕРВІСІВ ДЛЯ ОБРОБКИ ГЕОІНФОРМАЦІЙНИХ ДАНИХ

Метою розробки є реалізація модуля програмного продукту для керування геоінформаційними даними з використанням мікросервісної архітектури (рисунок 1.1). [1].

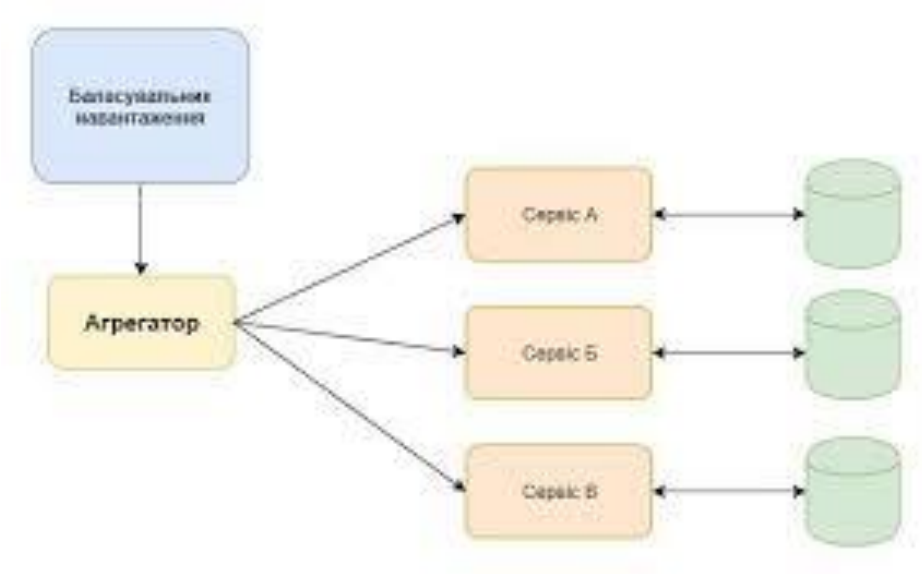

Рисунок 1.1 – Схема взаємодії мікросервісів між собою

Призначенням даного програмного засобу є надання програмного інтерфейсу який надає можливість за допомогою спеціальних пристроїв (пенетрометрів) зберігати різні виміри у системі. Прикладом таких даних є якість ущільнення грунту, характеристики щільності тощо. Іншим призначенням програмного продукту є надання чіткого і зрозумілого користувацького інтерфейсу, який може використовуватись інженерами та науковцями для оброки, аналізу.

Необхідними можливостями, які має забезпечувати система, є:

— можливість збереження геоінформаційних даних в системі;

— авторизація та аутентифікація користувачів в системі;

— можливість керування пристроями, яки використовуються для збору даних;

— можливість створення полів вимірів на карті;

— можливість редагування та обробки зібраних даних;

— можливість збереження зібраних даних на зовнішній пристрій;

— можливість завантаження даних з зовнішнього пристроя;

— можливість пошуку даних за критеріями;

— можливість створення груп даних;

1.1. Вимоги до архітектури програмного продукту

Однією із основних проблем при збереженні та обробці даних з реальних пристроїв є високе навантаження на систему через велику кількість подій, які відбуваються на пристрої.

Таким чином розроблювана система повинна відповідати деяким критеріям. А саме: повинна бути відмовостійкою, щоб мати можливість зберігати дані протягом всього часу роботи пристрої, повинна мати можливість горизонтального масштабування для рівномірного розподілення навантаження між декількома серверними машинами, повинна вміти працювати з даними одразу з декількох джерел.

Мікросервісна архітектура надає багато можливостей для реалізації наведених вище задач, а саме:

— можливість балансування навантаження між різними мікросервісами;

— сервіси запускаються швидко, що дозволяє змінювати конфігурацію сервісів в залежності від потреб навантаження;

— є можливість відокремленого масштабування кожного сервіса;

— кожен мікросервіс може бути створений з використанням технології, яка краще підходить для вирішення конкретної, більш атомарної задачі;

— можливість покращення сховища данних та апаратних ресурсів для найбільш навантажених модулів системи;

Наступною проблемою розробки системи є часті зміни різних бізнес правил для обробки даних, додавання нових типів пристроїв в систему з використанням нових протоколів передачі даних. Всі ці проблеми спонукають розробників системи до більш частих оновлень та перезавантаження окремих модулів системи.

В такому випадку мікросервісна архітектура надає користувачам цілий ряд переваг:

— можливе розгортання кожного сервіса окремо. Тож при змінах хоча б в одному із них, у вас є можливість розгортання цих змін, без враження інших, які продовжать працювати далі;

— вся система продовжить працювати у випадку неполадки в одному чи декількох модулях;

— організація в мікросервісах, як правило, на багато краще, тому що кожен мікросервіс жорстко відокремлений від інших. Саме тому мікросервіси вважаються кращими для пітрімки, тесування та роботи з кодом;

— коли сервіси розділені, їх набагато легше перекомпонувати або переконфігурувати, у випадку виконання різних задач;

Мікросервісну архітектуру доцільно використовувати, якщо всю систему планується розгорнути в хмарі, оскільки вона ідеально підходить для задачі горизонтального масштабування і прекрасно взаємодіє з можливостями хмарного середовища.

1.2. Вимоги до інтерфейсів програмного продукту

Програмний продукт повинен мати 2 основних інтерфейси. Один – програмний, який буде використовуватись для взаємодії з різними геоінформаційними пристроями, такими як пенетрометри. Інший – зрозумілий

користувацький інтерфейс, який може бути використаний науковцями, аналітиками, інженерами та іншими користувачами, які планують користуватись продуктом.

Програмний інтерфейс повинний бути сумісним з програмним забезпечення більшості відомих пристроїв для збору геоінформаційних даних. Є багато способів створити такий інтерфейс. Наприклад, можна використати один із відомих мережевих протоколів для передачі даних, яких буде використаних поверх мережі інтернет. Прикладами таких протоколів є: TCP/IP, UDP, HTTP. Якщо буде вибраний протокол HTTP, то як надбудову над ним можна використати один найвідоміших архітектурних стилів – такі як: REST (Representation state transfer), SOAP (Simple object access protocol), RPC (Remote procedure call).

Формат даних, за допомогою якого можуть обмінюватись даними програмний додаток та інженерний пристрій також можуть різнитись. Одними з найбільш популярних є: JSON (JavaScript object notation), XML(Extensible Markup Language), XMPP (Extensible messaging and presence protocol).

Щодо користувацького інтерфейсу – він повинен відповідати всім наявним вимогам. В першу чергу інтерфейс повинен відповідати принципу дружності – тобто бути простим у використанні і готовим задовольнити всі потреби користувача. Інтерфейс повинен бути гнучким – має надавати можливість адаптувати його під конкретну задачу. Інтерфейс повинен бути адаптованим – забезпечувати простоту переходу користувача від виконання одних функцій до інших.

Висновки до розділу 1

Були проаналізовані поставлені до розробки задачі. Поставлено цілі, необхідні для реалізації програмного продукту. Основними перевагами системи є мікросервісна архітектура, яка має ряд переваг перед іншими підходами в розробці програмного забезпечення.

2. АНАЛІЗ ПРОБЛЕМИ РОЗРОБКИ МІКРОСЕРВІСНОГО ДОДАТКУ ТА ЙОГО РОЗМІЩЕННЯ В ХМАРНОМУ СЕРЕДОВИЩІ

2.1. Основні принципи мікросервісної архітектури

Розглянемо основні елементи мікросервісної архітектури. (Рисунок 2.1)

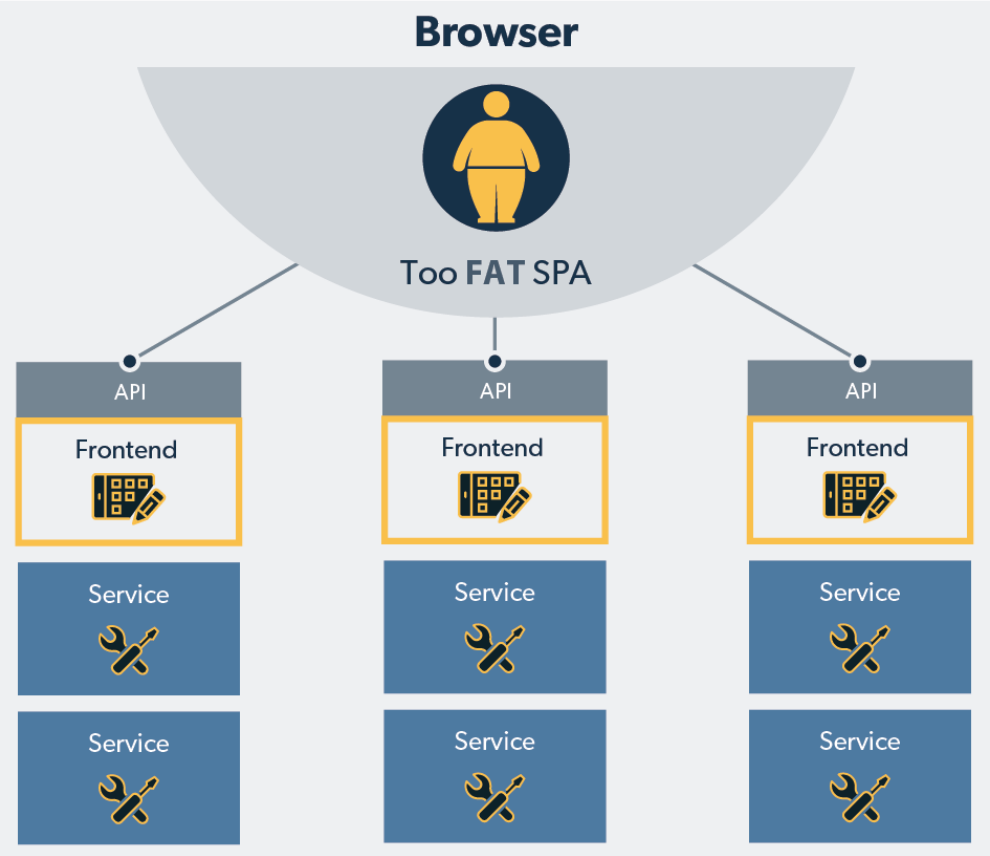

Рисунок 2.1 – Схема мікросервісної архітектури

Мікросервісний підхід або мікросервісна архітектура – це підхід, при якому вся система розбивається на невеликі компонентні частини, кожен з яких працює в окремому середовищі і взаємодіє з іншими з використанням різноманітних протоколів для обміну даними, такими як HTTP, TCP, брокери повідомлень тощо. Існує декілька підходів, за якими можна визначити спосіб розділення системи на

мікросервіси. Перший – побудова сервісів за доменною областю, тобто навколо певних програмних сутностей, які можна виділити в програмному продукті. Наприклад, сервіс полів, сервіс пристроїв, сервіс користувачів тощо. Інший підхід розбиває додаток на мікросервіси по бізнес-орієнованій схемі, тобто визначає цілі сервісу за потребами продукту.

Розглянемо основні проблеми, які можуть виникнути при розробці описаного програмного рішення. Так як додаток призначений для збору великої кількості даних з різних пристроїв – можна вважати його високонавантаженим та вирішувати багато задач пов'язаних з великим навантаження. Інша складність – велика кількість складових елементів системи, які мають високий рівень зв'язності між собою, що може викликати значні проблеми з підтримкою та постійним оновленням системи.

Розглянемо основні переваги мікросервісного підходу для вирішення описаних вище проблем геоінформаційної інженерії.

— **Масштабованість.** Однією з найбільших переваг підходу – є можливість горизонтального масштабування (Рисунок 2.2) різних модулів системи окремо один від одного і балансувати потужність для кожного персонально, на відміну від монолітного підходу, де нам потрібно масштабувати всю систему вертикально (Рисунок 2.3), якщо спостерігаються проблеми з навантаженням, навіть, однієї частини функціональності. Як було описано вище, геоінформаційна система може мати великі вимоги до навантаження;

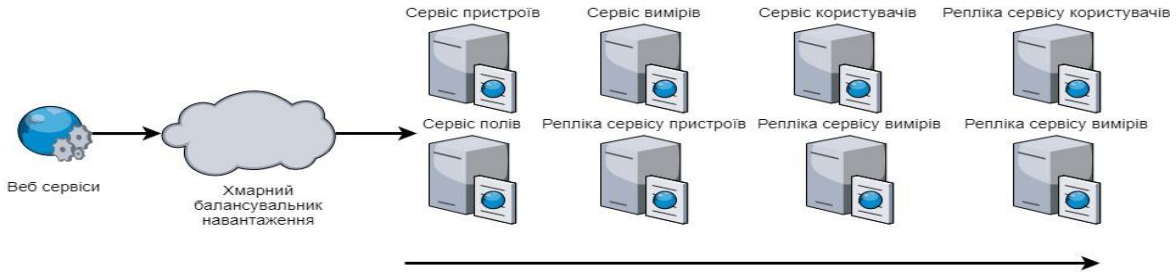

Рисунок 2.2 – Приклад горизонтального масштабування в мікросервісній архітектурі

— **Незалежне розгортання.** При використанні даної архітектури будь-які оновлення можуть вноситися в кожен модуль окремо та розгортати його незалежно

від інших компонентів системи. Це пришвидшує процес оновлення та додавання нових можливостей до системи, що є важливим для системи, яка працює з різними типами даних.

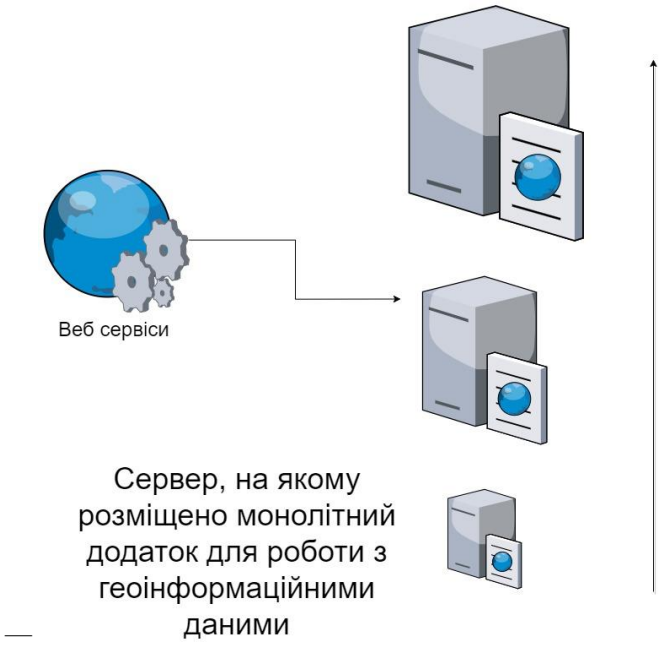

Рисунок 2.3 – Приклад вертикального масштабування в монолітній архітектурі

— **Гетерогенність технологій.** В області обробки великих масивів даних (якими виступають дані про події з різних пристроїв) дуже важливим є вибір технології під конкретну задачу. Наприклад, під одну задачу дуже важливо використати NoSQL базу даних для збереження великого об’єму гетерогенних даних, для іншої задачі набагато краще використати реляційну базу даних. Також це стосується продуктивності. Під одні задачі можна використати біль продуктивну мову програмування, наприклад, C++, для іншої більш важлива стабільність і консистентність, тому можна вибрати, наприклад, Java.

— **Стійкість системи.** В будь-якій, навіть найбільш найдійній системі можуть виникнути програмні або технічні несправності. І якщо у випадку моноліту в разі таких несправностей нам доведеться перезавантажувати всю систему, то у випадку мікросервісів ми можемо локалізувати проблему до конкретного модуля.

— **Модульність.** В системі, яка оперує великою кількістю сутностей можна, навіть, не помітити, як всі сутності перемішуються одна з одною. І з плином часу і розвитком системи вона може стати повністю непідтримуваною. Мікросервіси ж дозволяють фізично розділити модулі між собою, що дає можливість зменшити зв'язність і запобігти описаній вище проблемі.

Для вирішення програмних задач, при побудові програмного додатку існує безліч архітектурних рішень і дуже важливо максимально ретельно підійти до вибору архітектури. З наведених вище аргументів можна зробити висновок, що монолітний підхід непогано підходить для невеликих додатків, які вирішую прості задачі. Але при побудові додатку для вирішенні геоінформаційних інженерних задач одним із найкращих варіантів є застосування мікросервісної архітектури.

2.2. Аналіз існуючих рішень для розміщення мікросервісного додатку в хмарному середовищі

На сьогодні існує велика кількість готових програмних засобів для запуску та розміщення мікросервісів в хмарному середовищі. До таких засобів можна віднести:

CloudStack — спочатку розвивалась компанією Cloud.com і включала в себе ряд закритих компонентів доступних тільки для комерційної версії. Після поглинання Cloud.com корпорацією Citrix в червні 2011 року продукт було переведено в розділ повністю відкритих, а весь код було опубліковано під ліцензією GPLv3. В лютому 2012 року, Citrix випустили CloudStack 3 з підтримкою хмарного сховища Swift, яке розвивалося в рамках проекту OpenStack. Незабаром після випуску нової версії, в квітні 2012 року, Citrix передає CloudStack в Apache Software Foundation(далі ASF) і змінює ліцензію на Apache License 2 під якою проект розвивається і зараз. Версія CloudStack 4 була випущена уже під заступництвом ASF в листопаді 2012 року і є на даний момент стабільною версією.

Основними особливостями технології Cloud Stack є:

— Можливість роботи з все існуючими інтерфейсами, наприклад, Google Cloud API;

— Виділення і обмеження ресурсів може відбуватись автоматично;

— Можливе використання спеціального функціоналу для генерування звітів витрат;

— Інтерактивний Web-інтерфейс, створений з використанням найсучасніших технологій Web-розробким;

— Повна підтримка можливостей віртуалізації в мережах різного типу за допомогою ізолювання певних сегментів на окремі VLAN;

— Адаптивні ресурси, які можуть надаватися у відповідності до навантаження;

— Автоматизоване виділення місця для збереження файлів, різноманітних ресурсів, які використовує інфраструктура, можливість керування ресурсами в залежності від потреб балансувальника навантаження.

— Є можливість підключення зовнішніх розширень;

На сьогодні Cloud Stack це один з небагатьох продуктів, які можуть бути з легкістю розгорнуті і налаштовані без будь-яких складнощів. Практично весь функціонал, реалізований в Cloud Stack доступний з веб-інтерфейсу, в якому варто відзначити швидку роботу, відсутність зависань та помилок. Всі конфігурації доступні в розділі Global Settings, і не вимагає редагування конфігураційних файлів в терміналі Linux.

Eucalyptus Open Source — являє собою інфраструктуру для реалізації моделі хмарних обчислень рівня IAAS, ще одна платформа для побудови хмар. Свої Private Cloud, побудували на цій платформі, такі великі компанії як: Sony, Puma, NASA, Trend micro та інші. Існують 2 редакції Eucalyptus: платна і безкоштовна. У цих версіях дуже сильно відрізняється функціонал. Основною перевагою Eucalyptus є те, що його API повністю сумісний з Amazon API. Тобто, всі скрипти і програми, які

працюють з Amazon API, можуть бути використані і для нашого хмари, побудованого на Eucalyptus платформі.

Eucalyptus володіє наступними характеристиками:

— Інтерфейс, сумісний з EC2 і S3 (Webсервіси і інтерфейс Query/REST);

— Підтримка XEN і KVM;

— Простота інсталяції та розгорнення;

— Підтримка більшості дистрибутивів Linux (бінарні пакети та source-файли);

— Безпечна взаємодія компонентів з використанням SOAP та WS-security;

— Мінімальна модифікація Linux-оточення;

Eucalyptus — не найпростіша в використанні система, але один раз налаштувавши хмару ви отримаєте в розпорядження свій власний Amazon, який може працювати місяцями без зайвого втручання.

OpenStack — платформа с відкритим кодом для побудови хмар. У проект Openstack входить 3 продукту: Nova (аналог Amazon EC2), Swift (аналог Amazon S3), Glance (сервіс для надання образів). Програмний продукт OpenStack розроблений компанією Rackspace для комерційного надання "інфраструктури як сервісу" і для використання в наукових проектах NASA. Надалі вихідні коди були надані світовому незалежному спільноті розробників, і проект продовжив активно розвиватися, а також став доступний для безкоштовного приватного і комерційного використання. До розробки проекту OpenStack за останнім часом приєдналися більше 180 компаній. Вищий рівень членства («платиновий») в некомерційної організації у компаній:

— AT&T;

— Canonical;

— HP;

— IBM;

— Nebula;

— Rackspace;

— Red Hat;

— SUSE;

На світовому ринку компетенціями впровадження рішень на базі OpenStack володіють компанії Red Hat, Mirantis, Enovance, в Росії на 2014 рік про себе заявила компанія ASD Technologies.

Висновки до розділу 2

Було проаналізовано різні підходи до розміщення мікросервісів в хмарному середовищі та обрано середовище OpenStack як одне з найкращих через ряд переваг. OpenStack добре масштабується і здатна обслуговувати інфраструктуру з сотень тисяч віртуальних серверів.

3. ТЕХНОЛОГІЇ РОЗРОБКИ МІКРОСЕРВІСІВ ДЛЯ РОБОТИ З ГЕОІНФОРМАЦІЙНИМИ ДАНИМИ ТА РОБОТИ З ХМАРНИМ СЕРЕДОВИЩЕМ

Як мову програмування для розробки мікросервісів було обрано Java, як один із найпопулярніших засобів для розробки великих промислових додатків.

Основним середовищем розробки було середовище розробки від JetBrains IntelliJ Idea, що є одним із найпопулярніших середовищ розробки для Java, на які можна оформити студентську ліцензію.

Програмний засіб був розроблений з використанням можливостей об’єктно-орієнтовного підходу, принципів багатошарової MVC-архітектури на серверній частині з використанням фреймворку Spring Boot.

Як інструмент для асинхронного для обміну повідомленнями між сервісами була обрана Kafka.

Фреймворком для керування мікросервісами та інтеграцією до хмарного середовища було обрано Kubernetes.

3.1. «Спілкування» між мікросервісами в хмарному середовищі

Одним із найважливіших питань при побудові мікросервісної архітектури є протокол та формат обміну даними та повідомленнями між сервісами. Адже в сфері розробки ПО неможливо зробити систему де всі модулі на сто відсотків незалежні один від одного. Ось чому дуже важливо знайти правильну технологію для вирішення цієї задачі.

Існує два найбільш поширених способа комунікації мікросервісів між собою: синхронний (REST, RPC) та асинхронний обмін повідомленнями (ActiveMQ, RabbitMQ, Kafka).

Розглянемо переваги та недоліки синхронного підходу на прикладі архітектурного стилю REST.

У REST стилю (Рисунок 3.1) є безліч переваг, недоліків та обмежень, але ми сконцентруємо увагу на тих, які найбільш важливі в мікросервісній архітектурі. Одним із найбільш відомих понять в REST архітектурі є поняття ресурсу. Під ресурсом маються на увазі сутності, про які знає сервіс, наприклад, пристрій або вимір. Під час обробки запита сервер може згенерувати представлення, яке більш зручне для клієнта. Одними із найбільш поширених форматів для обміну даними є JSON та XML. Отримавши представлення ресурса клієнт може робити запити на його зміну і таким чином взаємодіяти з цим ресурсом. Дуже велику роль в REST-архітектурі грає протокол HTTP, адже цей протокол надає стандартні методи GET, POST, PUT, DELETE які дуже добре накладаються на приниципи REST. Наприклад, метод GET прийнято використовувати для отримання даних, POST для створення нового ресурсу, PUT для оновлення існуючого, а DELETE для видалення.

Одне, що бентежить людей, - це те, що REST та HTTP як правило використовуються разом. Але всесвітня мережа в більшості випадків і без того використуває HTTP, і не до кінця зрозуміло навіщо потрібен RESTful API якщо він робить те саме. Однак у обмеженнях REST немає нічого, що робить використання HTTP як протоколу передачі обов'язковим. Цілком можливо використовувати інші протоколи передачі, такі як SNMP, SMTP та інші, і ваш API все ще може бути RESTful API.

На практиці більшість - якщо не всі - API RESTful використовують HTTP як транспортний рівень, оскільки інфраструктура, сервери та клієнтські бібліотеки для HTTP вже широко доступні.

Однією з найбільших переваг описаного вище підходу є незалежність серверної реалізації від типу клієнта. Через один і той самий інтерфейс можуть спілкуватися з сервісом – веб-додаток, мобільний додаток, датчики для роботи з фізичними об’єктами та навіть інші мікросервіси в рамках однієї системи.

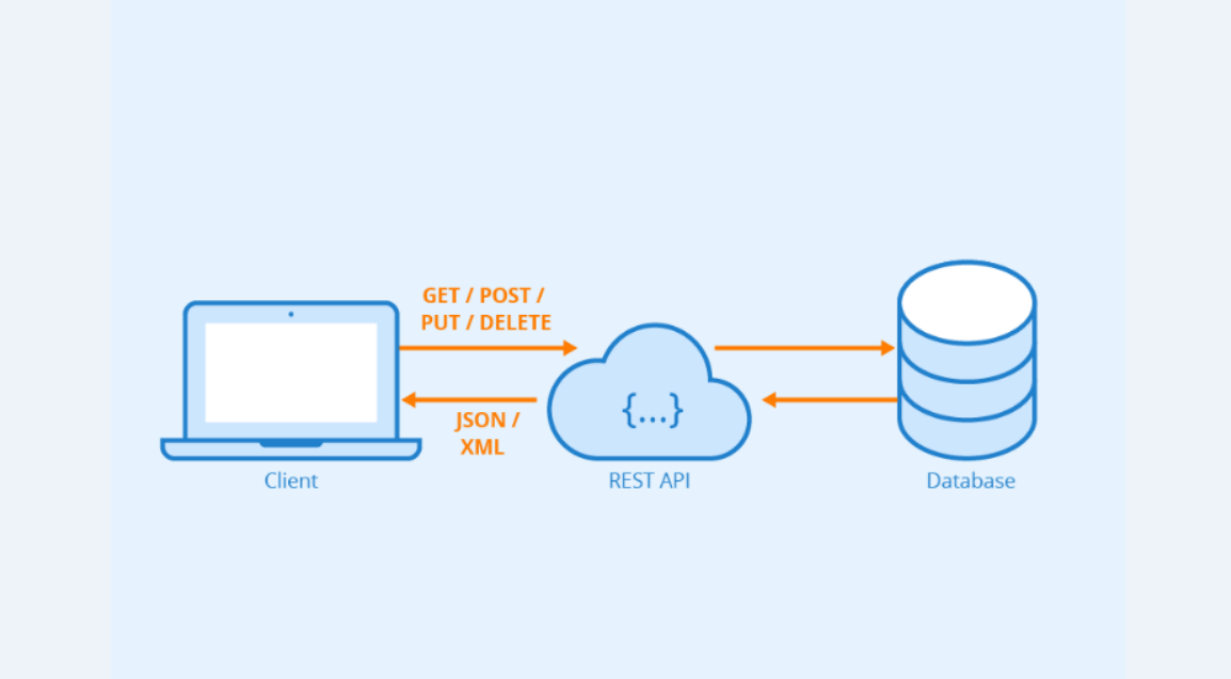

Рисунок 3.1 – Архітектурний стиль REST

Інший же спосіб для спілкування в розподілених системах – асинхронний обмін повідомленнями (Рисунок 3.2). Основними елементами в цьому підході є надання мікросервісами подій та способів визначення настання події іншими сервісами. Як правило брокери повідомлень, такі як Kafka або ActiveMQ намагаються охопити обидві задачі.

Генератори (producer) надають програмний інтерфейс для публікації подій або повідомлень до так званої черги, що представляє собою відокремлений сервер, основна задача якого отримувати, зберігати та передавати повідомлення отримувачам.

Споживачі (consumers) отримують інформацію при настанні певної події, про яку повідомив генератор. Брокери черги можуть обробляти стан споживачів, наприклад відстежувати повідомлення, які вони бачили раніше.

Повідомлення в чергу може відправляти будь-яка кількість генераторів. Так само будь-яка кількість споживачів може отримувати одне і те саме повідомлення. Таким чином ми можемо генерувати зв’язки один до багатьох в площині обміну

повідомленнями. Наприклад, про створення нового користувача може отримати інформацію одночасно і сервіс пристроїв і сервіс вимірів, але в рамках системи це буде лише одна подія.

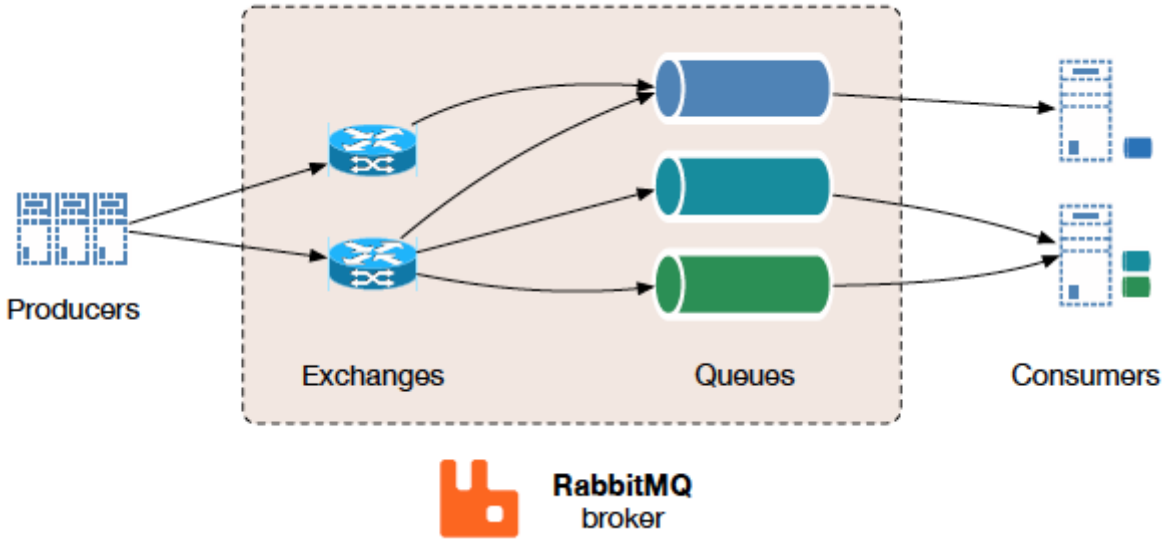

Рисунок 3.2 – Схема асинхронного обміну повідомленнями

Найбільш поширеними брокерами повідомлень є ActiveMQ, RabbitMQ та Apache Kafka. Для розробки даної системи було обрано систему Kafka, тому, що вона має певні переваги перед іншими.

— Вище було наведено пункти, щодо важливості масштабування даної системи. Kafka з самого початку позиціонувалась як розподілена система, яка може легко масштабуватись та розміщуватись в хмарному середовищі.

— Kafka має значні переваги у продуктивності в порівнянні з іншими системами.

— Kafka зберігає всі повідомлення на диску, що дуже важливо для такого пункту як гарантування доставки та відмовостійкість.

— Реплікованість. Навіть якщо один із модулів системи вийде з ладу, кожен topic має копії (репліки) і таким чином дані не можуть бути втрачені.

До основних компонентів системи Apache Kafka належать (Рисунок 3.3):

— Черга з повідомленнями певного типу з певною назвою, яка в терміноголії продукту називається topic. Повідомлення (message) які публікуються у чергу – це по факту певний набір даних у деякому форматі (XML, JSON, Avro), які можуть бути використані в певному бізнес процесі. Topic – це набір логічно пов’язаних між собою повідомлень.

— Producer (генератор) – це будь-який сервій, який публікує повідомлення в чергу.

— Всі повідомлення відправляються до кластеру серверів, який має назву broker, де вони зберігаються до моменти вичитування їх споживачем.

— Consumer (споживач) може підписатися на декілька топіків та отримувати повідомлення, як тільки вони потрапляють до брокера.

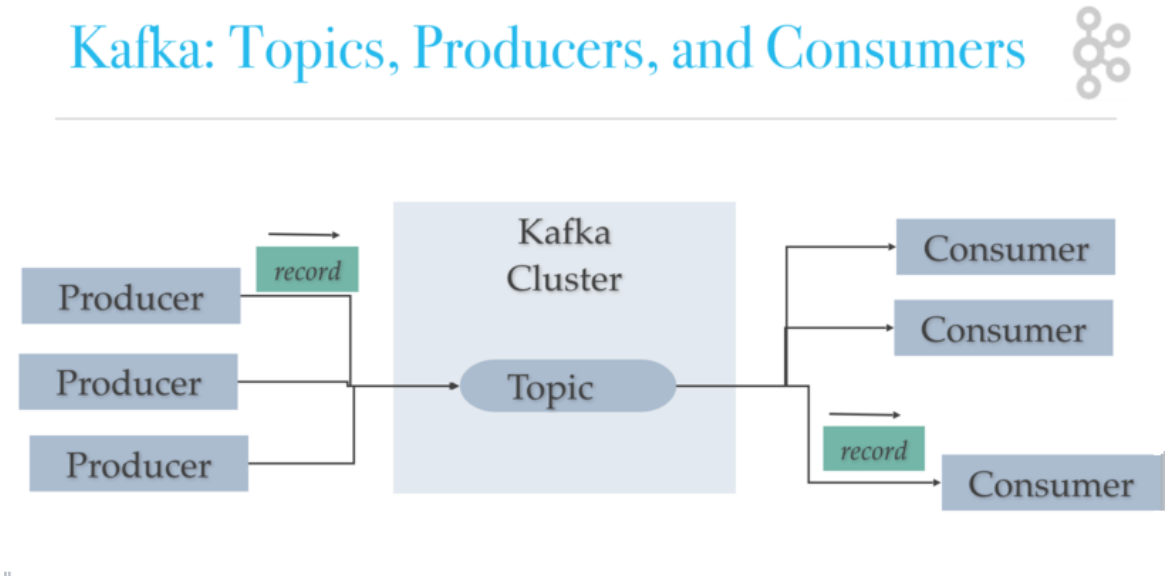

Рисунок 3.3 – Архітектури системи Apache Kafka

З самого початку Kafka проектувалась як система для роботи з великими обсягами даних. Її структура дозволяє легко керувати швидкістю обробки даних та продуктивністю при обчислення. Також існує механізм повторної обробки повідомлень, в разі, якщо споживач був тимчасово недоступний, що дозволяє запобігти помилкам в обчисленнях у разі технічних несправностей. Інтеграція

системи з розподіленим конфігуратором Zookeeper дозволяє швидко, безпечно та правильно працювати з будь-якими об’ємами даних. Таким чином, Kafka може бути використана для задач, де необхідна висока продуктивність, висока безпека та низьке використання ресурсів.

3.2. Програмні рішення для керування мікросервісами в хмарному середовищі

Однією з головних задач в розробці додатків з використанням мікросервісної архітектури є задача розміщення та керування великою кількістю серверних додатків в одному середовищі, налаштування масштабування, реплікації сервісів, а також пошук сервісів в одній мережі (service discovery). Задачі автоматичного розміщення, координації та управління великими комп’ютерними системами можна описати терміном – оркестрація. Оркестрація описує те, як сервіси повинні взаємодіяти між собою, використовуючи для цього обмін повідомленнями, включаючи бізнес-логіку та послідовність дій.

В основі будь-якої технології автоматизованої оркестрації лежить віртуалізація. Але, як відомо, будь-яка технологія віртуалізації, наприклад VirtualBox, має значний поганий вплив на продуктивність системи. Тому на фоні проблем віртуалізації почала розвиватись така область як контейнеризація.

Контейнери з’явились на корпоративному ринку не так давно. Багато чим це направлення з’явилось завдяки Docker – технології, яка дозволяє запускати додатки в контейнері, отримуючи результат, близький до звичайної віртуальної машини, але більш ефективний. Легковажність, зменшена ресурсоємність, практично повна незалежність від інфраструктури майже повністю забезпечили перехід на контейнери від звичайних віртуальних машин.

Розглянемо архітектуру системи Docker (Рисунок 3.4). Docker використовує архітектуру клієнт-сервер. Docker клієнт спілкується з так званим процесом Docker (демоном), задача якого – брати задачу створення, запуска та розподілення

контейнерів. Можна зауважити, що як клієнт так і сервер можуть працювати на одній системі, ви можете підключати клієнт до віддаленого docker процеса. Клієнт та сервер спілкуються через сокет або RESTful API.

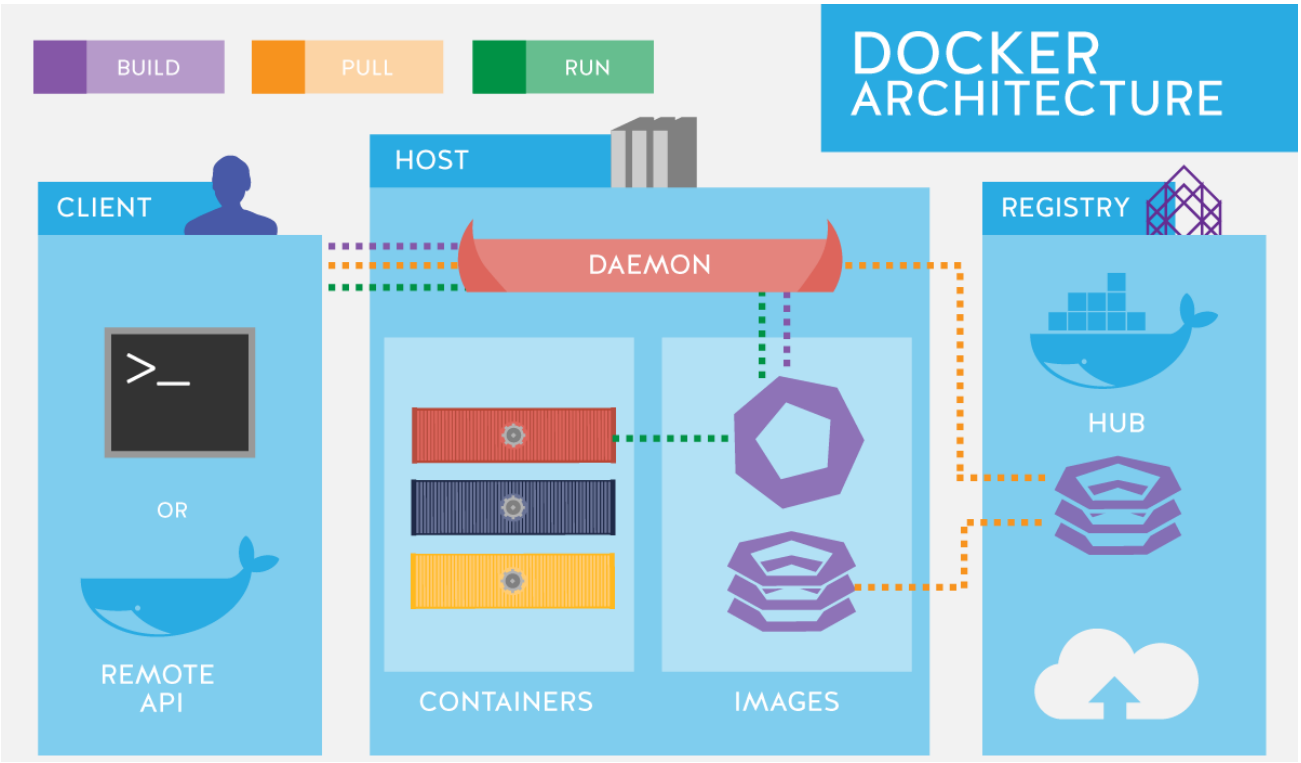

Рисунок 3.4 – Архітектури системи Docker

В основі кожного контейнера лежить так званий образ – це шаблон, з якого в подальшому створюється контейнер. Кожен образ складається із набору рівнів. Docker використовує union file system для зливання всіх рівнів в один образ. Union file system дозволяє файлам та директоріям різних файлових систем прозоро накладатись, створюючи когерентну файлову систему.

В основі кожного образа знаходиться базовий образ. Наприклад, Ubuntu, базовий образ Ubuntu, або Fedora, базовий образ дістрибутиву Fedora. Також у вас є можливість використання образів у вигляді бази для нових образів.

Ви можете створити нові образи Docker за допомогою базових образів, так звані інструкції є кроками для створення образів. Кожна із них може створити нові образи або рівні. Як інструкції можуть виступати наступні команди:

— Запуск певної команди.

— Робота з файлами в директорії (додавання, вилучення).

— Робота зі змінними оточення.

— Інструкції стартової точки входу в додаток.

Всі ці інструкції зберігаються в файлі Dockerfile. Docker зчитує ці інструкції, коли ви збираєте образ, виконує ці інструкції, та повертає кінцевий образ.

Розвиток платформ для оркестрації контейнерів – одне із напрямів ринкового сегменту систем віртуалізації. Вони служать для реалізації зручних та ефективних засобів розгортання контейнерних систем, побудови єдиної централізованої консолі для застосування політик керування. На сьогодні найбільш відомі такі системи, як Kubernetes, Docker Swarm та Apache Mesos. Крім них існують такі хмарні рішення, як Amazon EC2 Container Service або Microsoft Azure Container Service, але з точки зору популярності вони поступаються трьом фаворитам.

Три наведені технології були порівняні між собою в рамках вибору технології для розміщення мікросервісного додатку та вибраний один фаворит.

Kubernetes (Рисунок 3.5) – OpenSource-система для керування контейнерними кластерами. З’явилася в результаті в результаті напрацювань Google при використанні механізму для ізоляції процесів в віртуальному середовищі. В 2014 році Google відкрила код Kubernetes і стала поширювати систему під ліцензією Apache 2.0.

Завдяки тому, що Google відкрила код і зробила свою систему оркестрації відкритою, вона стала найпопулярнішою. Сьогодні багато людей розглядають її як найкраще та універсальне рішення для роботи з додатками в хмарному середовищі. Платформу оркестрації підтримують такі відомі вендори як Red Hat, IBM, CoreOS та інші.

Рисунок 3.5 – Логотип Kubernetes by Google

Kubernetes будує ефективну систему розподілення контейнерів по вузлам кластера в залежності від наявного навантаження та наявних вимог до роботи сервісів. Ця платформа оркестрації може обслуговувати величезну кількість хостів, запускати численні контейнери Docker та Rocket, відслідковувати їх стан, контролювати сумісну роботу, проводити балансування навантаження та інше.

Kubernetes – дуже складна система і дозволяє робити майже все, що відноситься до оркестрації контейнерів. Вона має дуже зручну панель приладів, гнучкі політики безпеки, вона підтримує розподілені мережеві файлові системи, має широкі можливості по визначенню метаданих для сервісів, що зручно у великих інфраструктурах. Із мінусів у Kubernetes можна виділити високу складність системи, що може скласти проблеми для недостатньо підготовлених спеціалістів та недостатність документації як для такої багатої на складові системи.

Docker Swarm (Рисунок 3.6) – друга за популярністю система оркестрації. Компанія Docker була першою, хто запропонував ефективну та зручну для корпоративного використання систему в 2013 році. Docker значно спростили розгортання повноцінних віртуальних систем та оркестрацію в цілому.

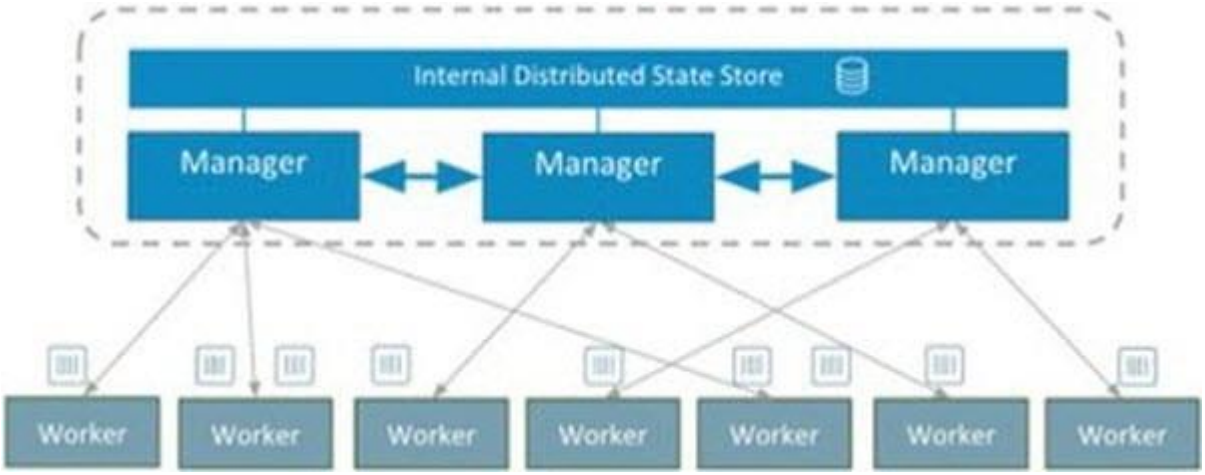

Рисунок 3.6 – Архітектура Docker Swarm

Інструмент контейнерної кластеризації Docker Swarm з’явився пізніше та став частиною платформи Docker. Він дозволяє об’єднувати Docker-хости в спільний віртуальний хост.

Docker Swarm цікава, насамперед, малим та середнім підприємствам, потреби яких не виходять за рамки запуску більше ніж 50 тисяч контейнерів і 1000 нод.

Не дивлячись на те, Docker Swarm займає більш скромні позиції в порівнянні з Kubernetes, її підтримка зі сторони корпоративного ринку достатньо вагома. Вона має значну підтримку зі сторони Microsoft Azure. Саме тому дана платформа зараз активно розвивається.

Із плюсів Docker Swarm можна виділити простоту роботи та швидкість освоєння. Якщо уже працював з Docker контейнерами, то навчитись працювати з Docker Swarm не складе значної складності.

Із мінусів – вузький функціонал, який не має таких значних можливостей відмовостійкості як у Kubernetes. Складність роботи з системою, якщо немає попереднього досвіду роботи з системами Docker.

Apache Mesos знаходиться на третьому місці за популярністю. Спочатку вона з’явилась як дослідницький проект в Університеті Берклі. Вперше вона була зібрана в повноцінний продукт та представлена публічно в 2009 році.

Apache Mesos (Рисунок 3.7) – це централізована відмовостійка система для керування кластером. Дозволяє об’єднувати в групи окрему вузли, згідно певних вимог, а також забезпечує ізоляцію від інших IT-ресурсів. Сутність роботи Apache Mesos відрізняється від звичної та уже традиційної моделі віртуалізації. Традиційний підхід передбачає розбиття обчислювального середовища, яке складається з багатьох фізичних машин на їх віртуальні аналоги. Apache Mesos працює інакше: він об’єднує існуючі об’єкти в єдиний віртуальний ресурс, формуючи крупні кластери та ефективну систему керування серверною інфраструктурою, в якій кожному кластеру виділяється свій індивідуальний пул ресурсів.

Система з підтримкою оркестрації отримала широку підтримку вендорів. В якості операційної системи для центрів обробки даних її використовують більше ніж шістдесят великих компаній, в тому числі Microsoft, Cisco, Dell, HPE, Autodesk, Twitter, eBay та інші. Apache Mesos буде цікавою великим компаніям, яким доводиться працювати з великою кількістю контейнерних кластерів, (на відміну від, наприклад, Docker Swarm).

В результаті порівняння для розміщення мікросервісів для обробки геоінформаційних даних було обрано систему Kubernetes, як систему, яка вміє запускати велику кількість контейнерів, при цьому надає значні можливості з масштабування та реплікації. З точки зору розробки Kubernetes має достатньо зручний інтерфейс взаємодії з користувачем для того, щоб бути використаним в розробці.

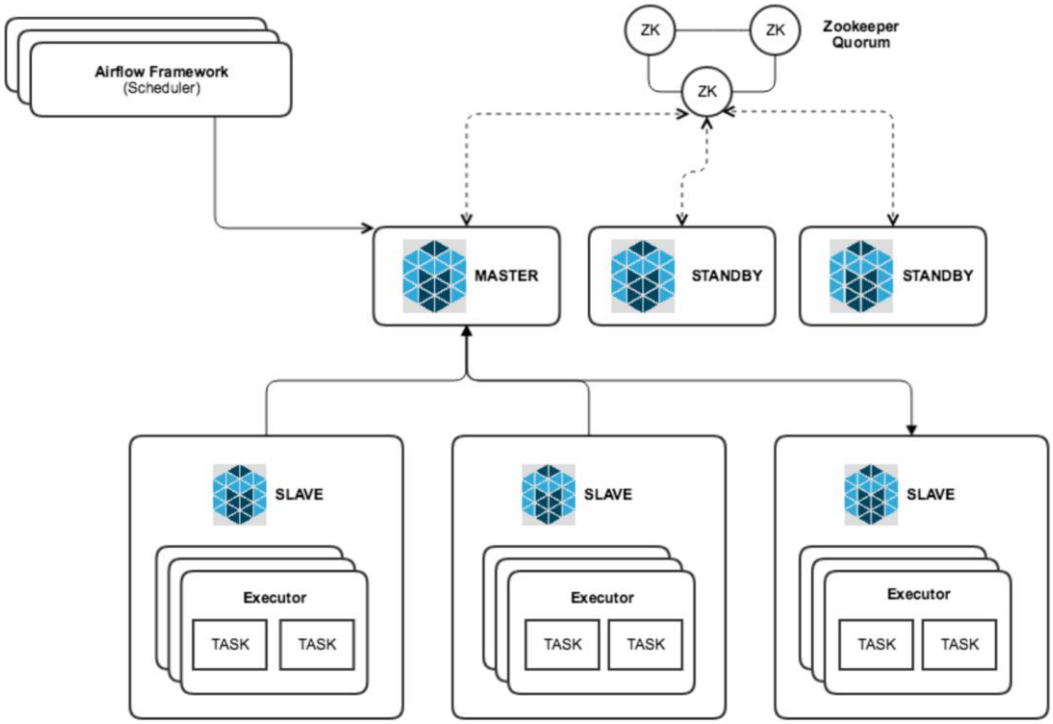

Рисунок 3.7 – Архітектура Apache Mesos

3.3. Мова програмування та середовище розробки

Однією із переваг мікросервісної архітектури є вільний вибір мови, яка буде використовуватись при розробці додатку, та навіть дозволяє комбінувати декілька мов в різних сервісах між собою. Але все ж таки існують мови, які більше підходять для вирішення конкретних задач. Для розробки додатку для обробки геоінформаційних даних було обрано мову програмування Java через ряд причин.

Java – мова програмування загального призначення, яка слідує парадигмі об’єктно-орієнтованого програмування та підходу «Написане раз може працювати де завгодно». Java дуже популярна серед мережевих та корпоративних додатків.

Java – це не тільки мова програмування, але і екосистема інструментів, яка охоплює майже все, що може знадобитись при програмуванні. В неї входять:

— **Java Development Kit –** комплект розробника Java. З його допомогою можна писати, компілювати та запускати код на Java.

— **Java Runtime Environment** – середовище виконання Java. Механізм розповсюдження програмного забезпечення, складається із автономної віртуальної машини Java, стандартної бібліотеки Java та інструментів конфігурування.

— **Integrated Development Environment –** інтегроване середовище розробки. Інструменти, які допомагають запускати, редагувати та компілювати код. Найбільш популярні – IntellijIDEA, Eclipse, NetBeans.

Розглянемо основні переваги Java:

Java включає в себе об’єктно-орієнтоване програмування – концепцію, в якій ви не тільки визначаєте тип даних та його структуру, але і набір функцій, застосовуваних до нього. Таким чином, структура даних стає об’єктом, яким можна керувати для створення відношень між різними об’єктами. Коли ви використовуєте процедурний підхід, у вас є лише можливість писати інструкції, визначати змінні або функції. ООП ж надає можливість групувати ці сутності з використанням контексту, маркувати кожне та посилатися на них в для окремого об’єкта.

В чому основні переваги ООП:

— При ООП можна повторно використовувати об’єкти в інших програмах.

— ООП передбачає помилки, оскільки об’єкти ховають інформацію, до якої не має бути доступа.

— ООП більш ефективно організовує структуру програм, у тому числі великих.

— ООП спрощує модернізацію та обслуговування старого коду.

Java створювалась як мова для розподіленого програмування: вона має вбудований механізм спільного використання даних та програм декількома комп’ютерами, що збільшує продуктивність та ефективність праці. В інших мовах потрібно використовувати зовнішній API для розподіленого обміну даними. В Java ця технологія вбудована. Специфічна для Java методологія розподілених обчислень називається Remote Method Invocation (RMI). RMI дозволяє використовувати всі переваги Java: безпека, незалежність від платформи та об’єктно-орієнтоване

програмування для розподілених обчислень. Крім того, Java також підтримує програмування сокетів та методологію розподілення CORBA для програмного обміну об’єктами між програмами, написаними різними мовами.

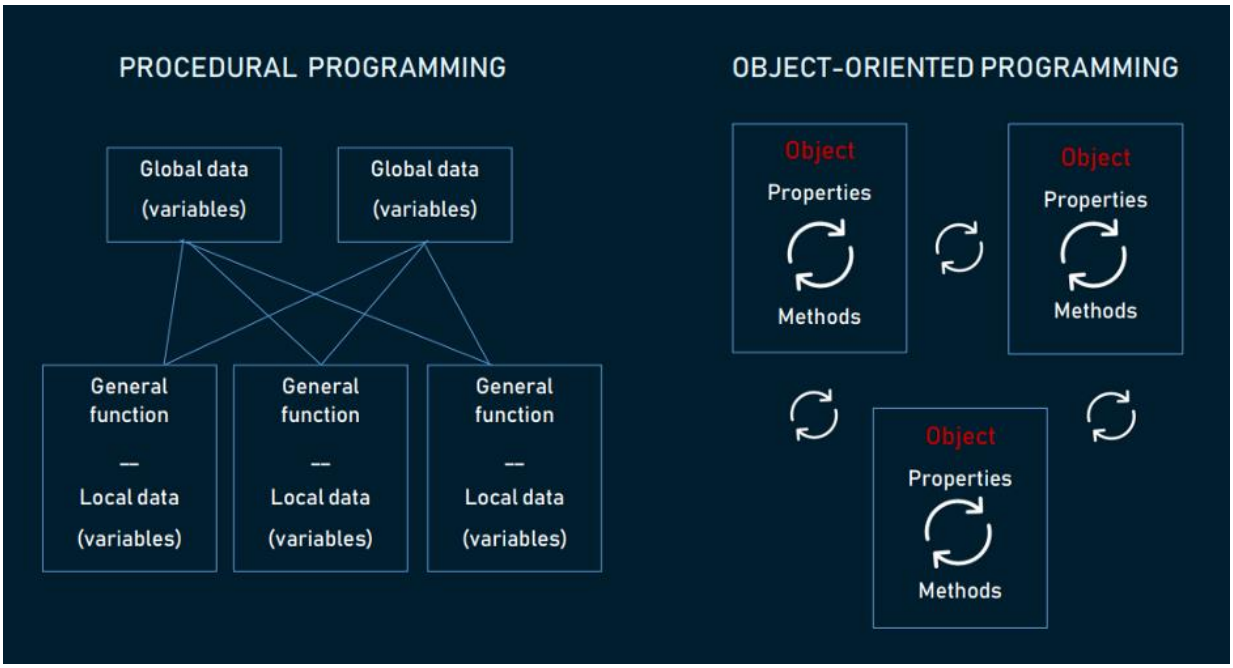

Рисунок 3.8 – Порівняння об’єктно-орієнтованого та процедурного підходів

Дуже важливим для великих розподілених додатків є керування пам’яттю. Розробникам Java не потрібно вручну писати код для керування пам’яттю завдяки автоматичному керуванню пам’яттю (АММ). Ефективність програми напряму пов’язана з пам’яттю. При цьому об’єм пам’яті обмежений. При написанні додатків на мовах з ручним керуванням пам’яттю, розробники ризикують забути виділити пам’ять, що приведе до збільшення об’єму займаної додатком пам’яті та проблемам з продуктивністю. Програми очистки пам’яті шукають об’єкти, які більше не використовуються програмою, та видаляють їх. Це впливає на роботу процесора, але розумна оптимізація та конфігурація дозволяють знизити цей вплив.

Мова Java має дуже добре спроектований механізм роботи з багатопоточністю, і вважається одним із найкращих серед всіх мов програмування. Потік – це найменша одиниця обробки в програмуванні. Для того, щоб максимально

ефективно використовувати процесорний час, Java дозволяє запускати потоки одночасно, що називається багатопоточністю.

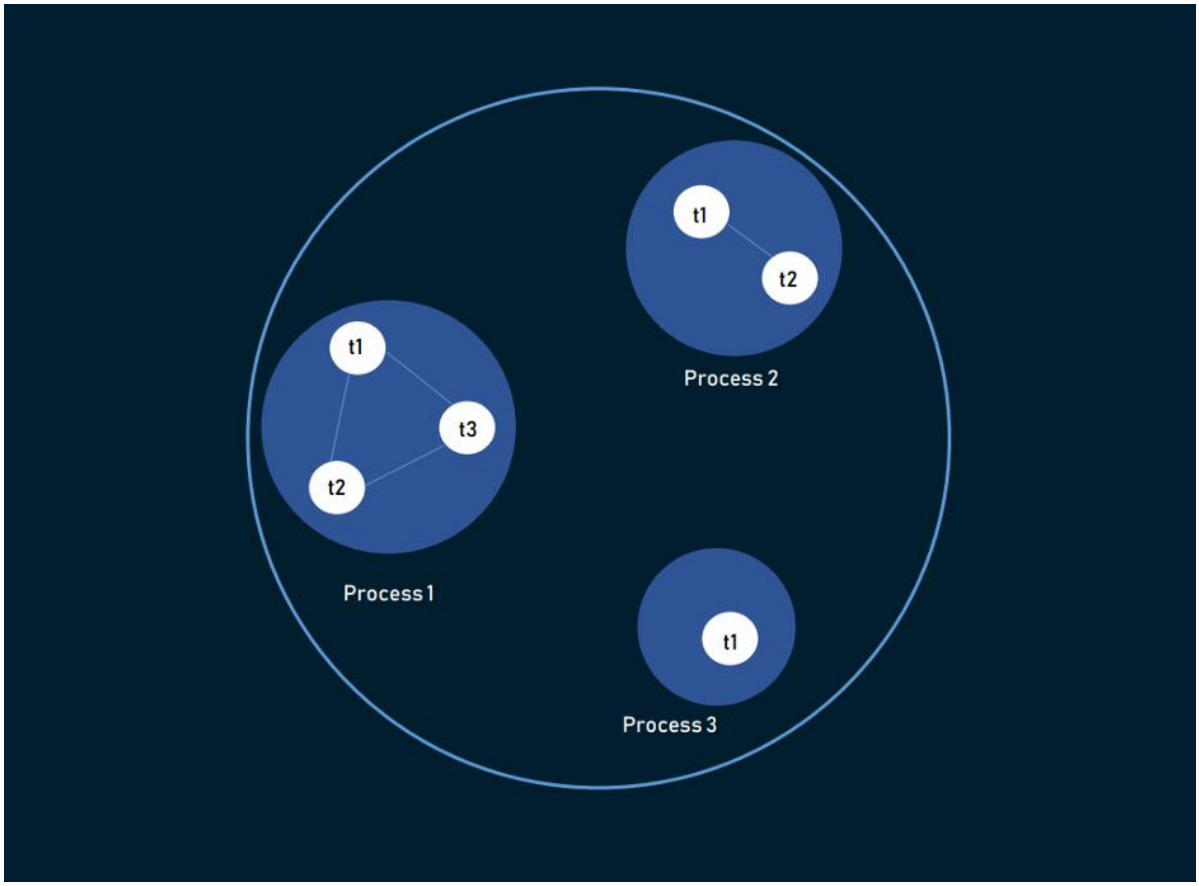

Рисунок 3.9 – Модель багатопоточності Java

Потоки використовують одну і ту саму область пам’яті, тому між ними можна швидко переключатися. Потоки незалежні один від одного: один потік не впливає на роботу інших потоків. Це особливо корисно в великих навантажених серверних додатках.

3.4. База даних для мікросервісів

Для будь-якого великого додатку дуже важливим є питання вибору бази даних. В геоінформаційному додатку дуже велика кількість даних буде зберігатися в базі даних, тому питання вибору найкращої є дуже важливим. Для даного проекту

було прийнято рішення – використовувати базу даних PostgreSQL через ряд переваг цієї системи.

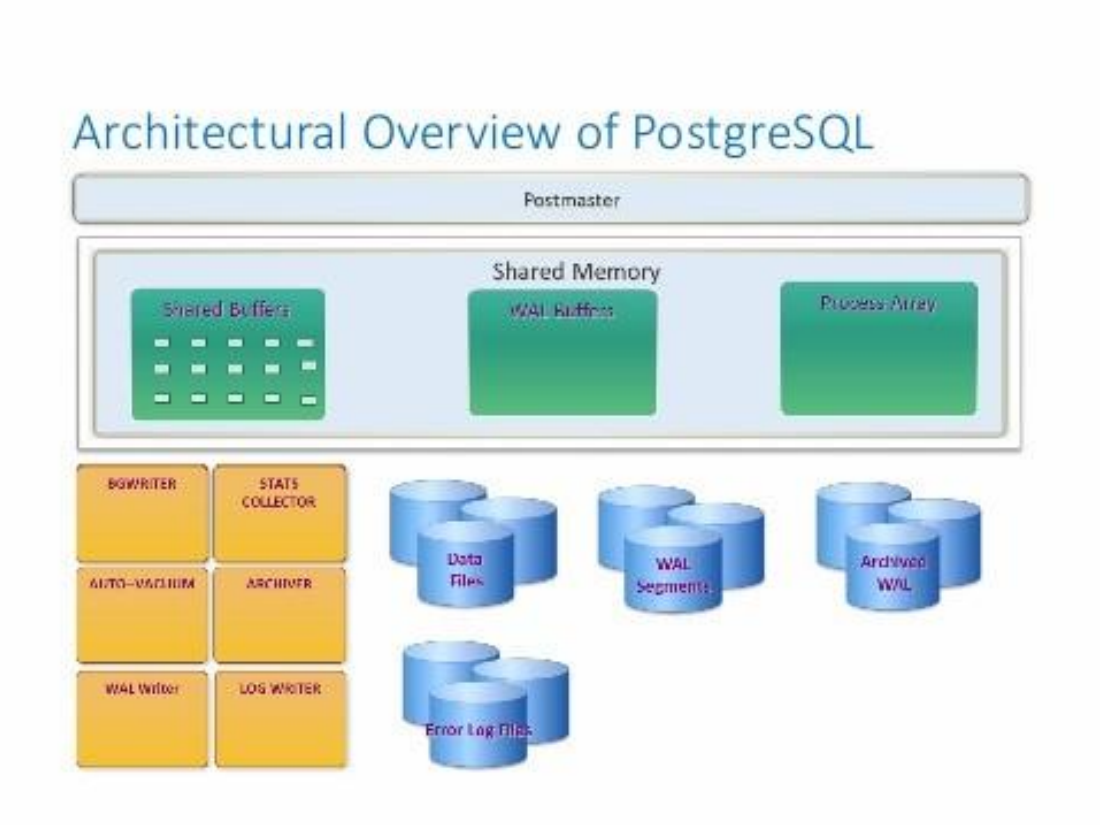

Рисунок 3.10 – Архітектура БД PostgreSQL

Тільки PostgreSQL забезпечує продуктивність та функції корпоративного класу серед поточних СУБД з відкритим кодом без кінця можливостей розробки. Також користувачі PostgreSQL можуть безпосередньо брати участь у спільноті та публікувати та ділитися незручностями та помилками.

PostgreSQL має дуже важливу функцію SQL під назвою "Збережені процедури", які можна використовувати для серверного середовища. Крім того БД підтримує мови, схожі на PL/SQL в Oracle, такі як PL/pgSQL, PL/Python, PL/Perl, C/C ++ та PL/R.

PostgreSQL підтримує ACID (Атомарність, Консистентність, Ізоляцію, Довговічність).

PostgreSQL надає не тільки методи індексу B+ дерева, але й різні види методів, такі як GIN (Узагальнений інвертований індекс) та GiST (Генералізоване дерево пошуку) тощо.

Повнотекстовий пошук доступний під час пошуку рядків із виконанням векторної операції та пошуку рядків є дуже важливим під час побудови алгоритмів пошуку.

PostgreSQL підтримує різноманітні методи реплікації, такі як Streaming Replication, Slony-I та cascading.

PostgreSQL підтримує різні види методів зберігання географічних даних, таких як PostGIS, Store Key-Value Store та DBLink, що є дуже важливим при роботі з геоінформаційними даними.

PostgreSQL надзвичайно багатий функціональними можливостями, з безліччю вбудованих можливостей і незліченною кількістю способів їх індивідуалізації та розширення для задоволення різних потреб. Це загальновизнана, надійна і зріліста БД, тому це рішення для баз даних варто зусиль будь-якого великого підприємства. При цьому він залишається доступним і ефективним також і для невеликих проектів.

3.5. Тестування мікросервісів.

Так як мікросервісна архітектура включає в себе велику кількість незалежних між собою модулів, це викликає певні складності з тестуванням. Тому тестування саме мікросервісої архітектури потребує додаткової уваги.

Поєднання архітектурного стилю мікросервісів та контейнерної інфраструктури вимагає стратегії тестування, сумісної з новими технологіями. Архітектура мікросервісу більше покладається на віддалену залежність і менше на компоненти, що перебувають у процесі, і ваша стратегія тестування та тестові середовища повинні адаптуватися до цих змін.

Більше мережевого зв’язку призводить до більших зусиль, витрачених на тестування з'єднань та контрактів між вашими мікросервісами. Також існує кілька нових методів тестування для обробки залежних компонентів під час переміщення до контейнерної інфраструктури, що часто виникає при розробці мікросервісів.

Основними категоріями рішень для тестування мікросервісів є ті, які вже доступні в монолітних архітектурах, але також застосовні до мікросервісів, і ті, що розроблені спеціально для архітектур мікросервісів.

Методи, доступні в монолітних архітектурах, використовують тестові подвійні елементи, такі як заглушки, макети або віртуальні сервіси; підключення до реальних тестових примірників резервних або сторонніх систем; і контрактне тестування.

Методи, доступні для архітектури мікросервісів (Рисунок 3.11), - це тестові контейнери, такі як контейнери для випробування баз даних, тестові контейнери для віртуалізації сервісу та тестові контейнери сторонніх служб (наприклад, тестовий контейнер Kafka, тестовий контейнер PostgreSQL або тестовий контейнер для віртуальних пристроїв).

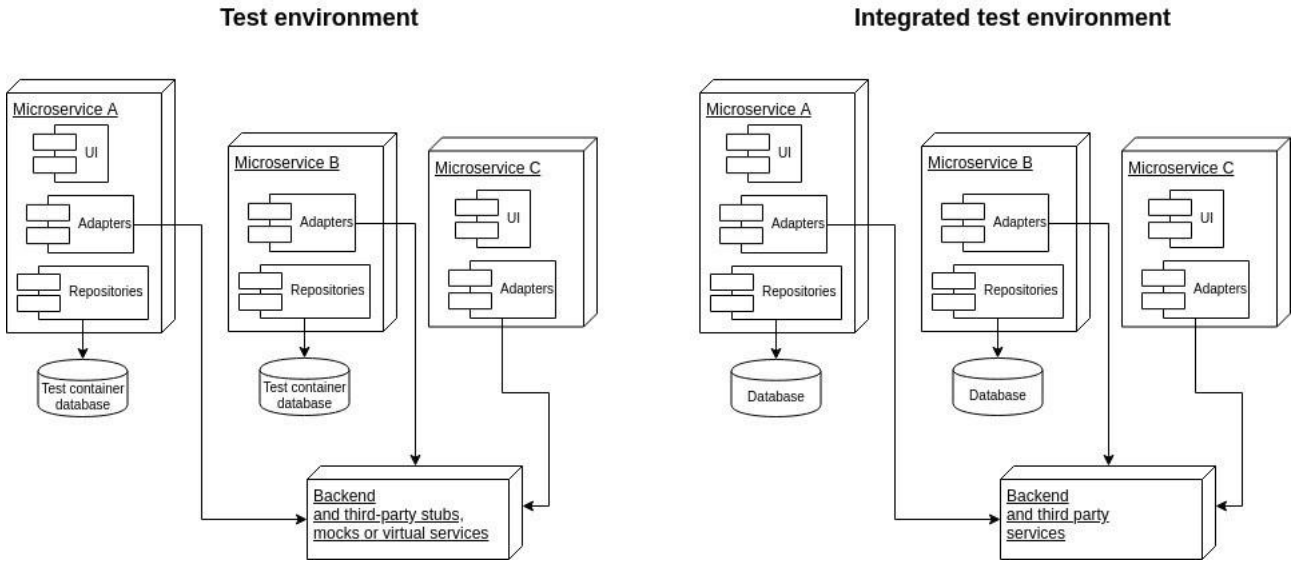

Рисунок 3.11 – Схема тестового середовища в мікросервісній архітектурі

Не зважаючи на те, що в мікросервісах використовується багато методів, які використовуються для монолітних архітектур, а також нові методи, що включають

контейнери, придатність різних методів тестування може змінюватися, оскільки петлі зворотного зв’язку в архітектурі мікросервісів є більш жорсткими, оскільки команди, як правило, розподілені та перехресно функціональні.

Поширена тема в перерахованих вище ресурсах полягає в тому, що вам потрібно керувати залежними компонентами для тестування мікропослуг економічно та ефективно. Залежно від ваших потреб, ви можете вибрати один із варіантів, перелічених вище, або їх комбінацію.

Тестове середовище для тестування мікросервісів може використовувати реальні залежності замість тестових заглушок.

Є кілька типів компонентів, з якими ваш мікросервіс може спілкуватися в рамках тестового сценарію:

— Ви можете протестувати мікросервіс за допомогою тестового примірника іншого мікросервісу. Наприклад, під час тестування мікросервісу А ви підключаєте його до тестового мікросервісу Б і тестуєте їх разом

— Ви можете протестувати мікросервіс із виробничим екземпляром іншого мікросервісу. Наприклад, під час тестування мікросервісу А ви підключаєте його до реального екземпляра мікросервісу В і протестуєте їх разом, перш ніж випускати мікросервіс А у реліз.

— Ви можете протестувати мікросервіс із сторонніми залежностями. Наприклад, під час тестування мікросервісу А ви підключаєте його до реального екземпляра сторонньої системи.

— Ви можете протестувати мікросервіс із застарілими монолітніми внутрішніми залежностями. Наприклад, під час тестування мікросервісу А ви підключаєте його до тестового примірника монолітної системи.

— Ви можете протестувати мікросервіс із непрограмними (апаратними) залежностями. Наприклад, під час тестування мікросервісу А ви підключаєте його до апаратного пристрою, відповідального за виконання функції.

Далі перерахуємо тестові компоненти, які можна використовувати у ваших тестах мікросервісу замість перелічених вище. Звичайно, ви можете використовувати так звані тестові дублери у своїх тестах на мікросервіси, які прикидаються справжніми залежностями у рамках тестів. Ви можете вибрати кілька методів, залежно від типу залежності та проблеми:

— **Макет (mock)** замінює об'єкт, від якого мікросервіс залежить конкретним тестовим об'єктом, який перевіряє, що мікросервіс використовує його правильно.

— **Заглушка (stub)** замінює об'єкт, від якого залежить мікросервіс, специфічним для тесту об'єктом, який надає дані тесту до мікросервісу. Дані тесту можуть бути статичними або динамічними.

— **Симулятор** – це розумна версія заглушки, яка імітує частину поведінки системи, від якої залежить мікросервіс. Наприклад, замість підключення до реальної платіжної системи в тесті, ви можете підключитися до симулятора, який реалізує частину функції оплати.

— **Віртуалізація служби** також називається моделюванням API або макетом API. Це практика заміни реальних залежних компонентів тестовими версіями, створеними за допомогою потужних інструментів для віртуалізації служб. Інструменти для віртуалізації сервісів дозволяють набути подібного імітатора досвіду, але з меншими зусиллями розробників та тестувальників. Замість того, щоб створювати індивідуальний тестовий дублер на залежність, інструменти, що не є частиною сервісу, дбають про шаблонну функціональність, яка є загальною для типових реалізацій. Інструменти віртуалізації послуг зазвичай пропонують більше функцій, ніж заглушки або макети, як запити запитів і відповідей, або вбудована підтримка багатьох технологій, таких як HTTP, JMS, FTP або gRPC.

— Можна використовувати базу даних в пам'яті для заміни реального примірника бази даних для тестування.

— Ви можете запустити тестовий контейнер, тестовий екземпляр залежності на збірку або конвеєр всередині контейнера, замість того, щоб використовувати

екземпляр, спільний для багатьох команд, збірок або конвеєрів. Наприклад, ви можете використовувати тестовий контейнер бази даних або тестовий контейнер для віртуалізації служби.

— Ви можете використовувати «legacy in box» підхід. Замість того, щоб покладатися на спільне тестове середовище, ви можете запустити застарілу систему в контейнері. Ви можете налаштувати його таким чином, що відповідає вашим потребам тестування.

Тестування контрактів є найважливішим елементом при тестуванні слабко пов'язаних компонентів, таких як мікросервіси.

Контракт описує, як компоненти спілкуються та взаємодіють один з одним - як формати повідомлень, які використовуються між компонентами, так і поведінкові очікування компонентів. Ви використовуєте контрактне тестування, щоб переконатися, що контракти між компонентами виконуються; це дає вам впевненість у тому, що компоненти можуть працювати разом. Коли ви використовуєте компоненти, що залежать від тесту (такі як тестові дублери), ви також можете використовувати контрактне тестування, щоб переконатися, що вони виконують останню або будь-яку конкретну версію контракту. Ось кілька способів тестування або управління контрактами між компонентами:

— Під час створення тестового контракту ваш тестовий параметр являє собою знімок контракту між компонентами в певний момент часу. Цей знімок може застаріти. Ви можете перевірити знімки контрактів в автоматизованому вигляді. Ви можете запустити тестовий контейнер, тестовий екземпляр залежності на збірку або конвеєр всередині контейнера, замість того, щоб використовувати екземпляр, спільний для багатьох команд, збірок або конвеєрів. Наприклад, ви можете використовувати тестовий контейнер бази даних або тестовий контейнер для віртуалізації служби.

— Оновлення знімків у контракті дозволяє перезаписати (оновити) контракти між компонентами. Як правило, оновлення буде обслуговувати синтаксис і частково семантику контракту.

— Тестування контрактів, орієнтоване на споживачів, є окремою стратегією тестування мікросервісів. Договори, орієнтовані на споживачів, можна розділити на виробника (producer) та споживачів (consumer). Тестування контрактів, орієнтоване на споживачів, підтверджує, що виробник надає договір, який відповідає усім очікуванням споживачів. Споживачі перевіряють, що виробники все ще надають структуру повідомлень та поведінку, яка їм потрібна.

— Тестування контрактів інтеграції за контрактом може перевірити контракт між модулем з'єднання у вашому мікросервісі та залежним компонентом. У цьому випадку контракт, як правило, більше орієнтований на виробників, а не на споживача.

— Мікросервіс може бути меншим за визначенням, але за допомогою тестування одиниць (unit test) ви можете протестувати сервіс більш детально. Тест одиниці фокусується на найменшій частині тестованого програмного забезпечення, щоб з’ясувати, чи працює цей компонент як слід

— Використовуйте тестування контрактів для незалежних релізів компонентів, якщо ви хочете самостійно випустити два залежні компоненти. Ви повинні пам’ятати, що треба перевіряти комбінації останніх релізних артефактів.

— Кінцеве тестування (E2E) означає перевірити, що всі компоненти добре працюють разом з точки зору користувача. Це означає, що контракти між компонентами неявно підтверджуються під час виконання тестів кінцевих тестів споживача.

Висновки до розділу 3

Отже із проаналізованих технологій було вибрано основні, які підходять для реалізації даного програмного додатку. Основними технологіями є: мова

програмування Java, яка використовується для розробки програмного коду всіх мікросервісів, фреймворк для керування мікросервісами Kubernetes, технологія для контейнеризації Docker. Для збереження даних було обрано базу даних PostgreSQL.

Також було обрано стратегії тестування мікросервісів, так як це є важливою частиною розробки системи за цією методологією.

4. ОПИС СИСТЕМИ ДЛЯ ЗБЕРЕЖЕННЯ ТА ОБРОБКИ ГЕОІНФОРМАЦІЙНИХ ДАНИХ ТА РОБОТИ З ХМАРНИМ СЕРЕДОВИЩЕМ

У цьому розділі розглядаються основні аспекти при розробці програмного інтерфейсу, хмарного середовища, а також в ньому проілюстровані діаграми класів і послідовностей для повного розуміння структури сервісу та роботи з хмарним середовищем.

4.1. Взаємодія з хмарним середовищем за допомогою фреймворку для оркестрації мікросервісів Kubernetes.

Фреймворк Kubernetes – це головна технологія, яка дозволяє розміщувати мікросервіси в хмарному середовищі, автоматично керує навантаженням між сервісами, створює нові ноди в хмарному середовищі, таким чином виконуючи задачу масштабування. Kubernetes виступає балансувальником навантаження і вміє автоматично розподіляти трафік між всіма репліками одного мікросервісу. Kubernetes дуже добре виконує задачу service discovery, яка необхідна для роботи мікросервісного додатку (принципи роботи service discovery наведені нижче). Розглянемо основні складові Kubernetes:

— **Nodes**. Нода – це основна складова, комп’ютер в кластері Kubernetes.

— **Pod**. Pod – це набір контейнерів, які мають спільні розділи, які запускаються атомарно, як єдине ціле.

— **Replication controller**. Replication controller контролює та забезпечує запуск pod в певний момент часу. Наприклад, якщо один под виключається через технічні несправності, replication controller його одразу ж створює знову.

— **Services**. Сервіси в Kubernetes це абстрактна функціональність, яка об’єднує pod у певні набори та визначає політику доступу до них.

— **Volumes**. Volume (розділ) – це файловий розділ з даними, доступ до якого можна отримати зсередини контейнера.

— **Labels**. Label – це маркери, формату ключ/значення, якими можна визначити об’єкти, наприклад, pod. Labels використовуються для створення наборів об’єктів.

— **Kubectl Command Line Interface**. Kubectl інтерфейс терміналу, через який користувач може взаємодіяти з системою.

Розглянемо архітектуру Kubernetes. Працюючий Kubernetes кластер працює з агентом, який запущений на kubelet нодах і компоненти майстра такі як APIs, scheduler, etc, поверх рішення з розподіленим сховищем. Наведена схема (Рисунок 4.1) показує бажаний, в кінцевому підсумку, стан, хоча все ще ведеться робота над деякими речами, наприклад: як зробити так, щоб kubelet (всі компоненти, насправді) самостійно запускався в контейнері, що зробить планувальник на 100%, підключаємим.

При погляді на архітектуру системи ми можемо розбити її на сервіси, які працюють на кожній ноді і сервіси рівня управління кластера. На кожній ноді Kubernetes запускаються сервіси, необхідні для управління нодою з боку майстра і для запуску додатків. Звичайно, на кожній ноді запускається Docker. Docker забезпечить регулярне завантаження образів і запуск контейнерів.

Kubelet керує pod'ами, їх контейнерами, образами, розділами, etc.

Також на кожній ноді запускається простий proxy-балансувальник. Цей сервіс запускається на кожній ноді і налаштовується в Kubernetes API. Kube-Proxy може виконувати найпростіше перенаправлення потоків TCP і UDP (round robin) між набором бекенд нод.

Система управління Kubernetes розділена на кілька компонентів. В даний момент всі вони запускаються на майстер-ноді, але незабаром це буде змінено для можливості створення відмов кластеру. Ці компоненти працюють разом, щоб забезпечити єдине уявлення кластера.

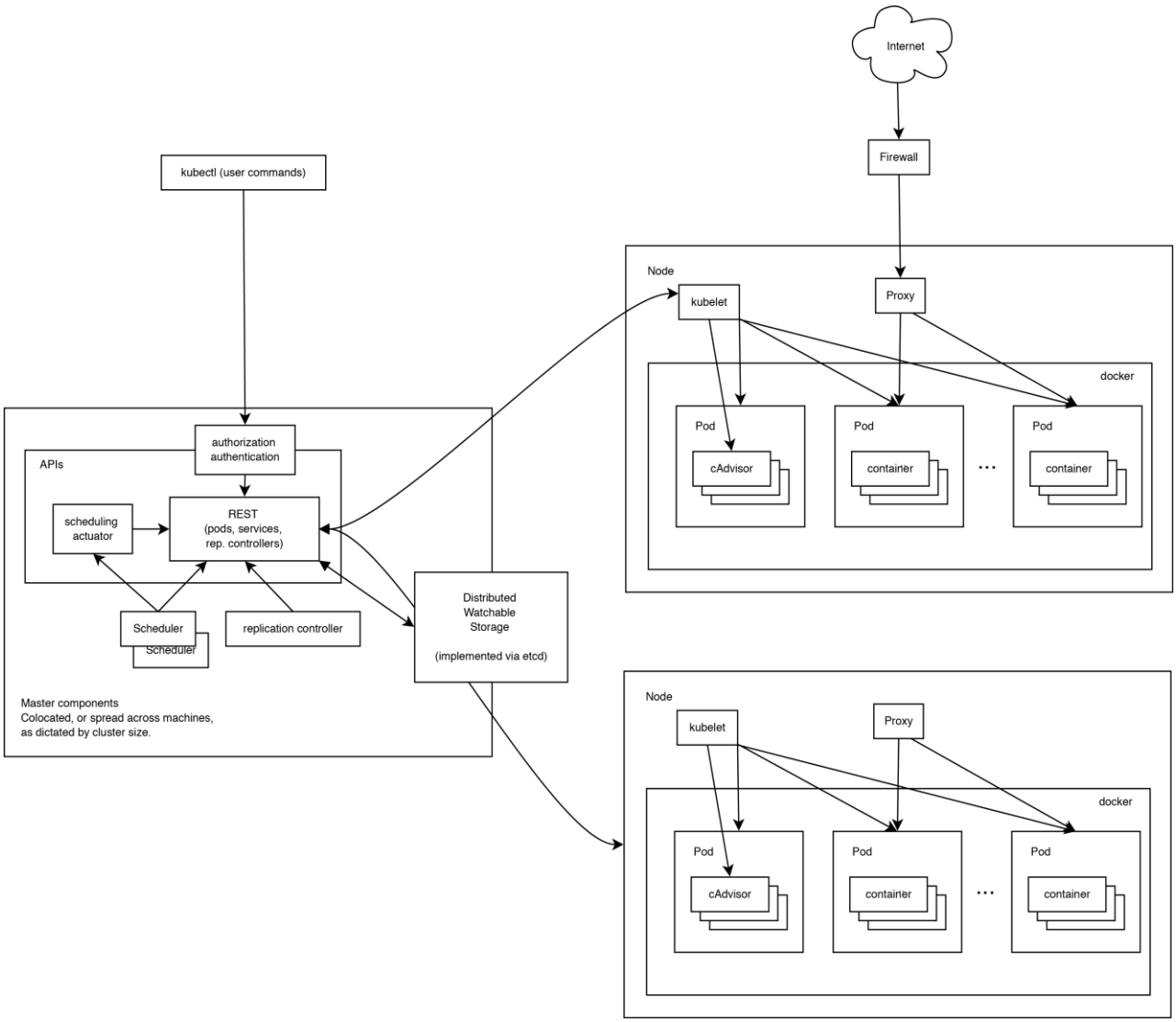

Рисунок 4.1 – Архітектура Kubernetes

Стан майстра зберігається в etcd екземплярі. Таким чином можна забезпечити правильне конфігурування та завчасне повідомлення елементів системи.

API Kubernetes надає можливість забезпечити серверну роботу API (програмного інтерфейсу). Його використовують для отримання функціональності CRUD та вбудованої бізнес-логіки, яка реалізована в відокремлених компонентах або в плагінах. Основною його задачею є обробка REST запитів, за допомогою їх перевірки і оновлення відповідних об'єктів etcd (і частково в інших сховищах).

Scheduler (планувальник) служить для планування задач за певний час і прив'язує незапущені pod'и до іменної ноди через виклик binding API. Scheduler підтримує множинні scheduler'и.

Всі інші функції рівня кластера представлені в так званому Controller Manager. Наприклад, машини виявляються, управляються і контролюються за допомогою node controller. Ця сутність в результаті може бути розділена на окремі компоненти, щоб зробити їх підключення незалежним. ReplicationController - це механізм, який базується на API pod.

Розглянемо приклад розміщення сервісу в Kubernetes. З точки зору користувача Kubernetes надає нам можливість роботи з декількома сутностями (Рисунок 4.2).

— **Deployment**. З точки зору розробки deployment є одиницею мікросервіса. Наприклад, в нашому випадку deploymentом може бути сервіс вимірів або сервіс пристроїв. Для того щоб створити deployment треба описати основну конфігурацію в файлі k8s.yml. Також цей файл обов’язково повинен містити посилання на docker image, який буде розміщено в кластері

— **Pod**. Набори контейнерів, які будуються для одного Deployment. У випадку з мікросервісами нашої системи – це репліки одного мікросервісу. Наприклад, для мікросервісу вимірів ми хочемо мати 3 репліки, щоб балансувати навантаження. Таким чином на один deployment measurement ми будемо мати три measurement pod.

— **Service**. Це логічна одиниця DNS в кластері, яка дозволяє іншими сервісам визначати адресу сервісу і взаємодіяти з ним. Кількість сервісів, як правило, дорівнює кількості deployment.

— **Load balancer** – механізм, який автоматично розподіляє трафік між репліками одного сервісу і дозволяє оновлювати сервіси без простоїв.

— **Replica set** – логічний набір реплікацій одного мікросервісу.

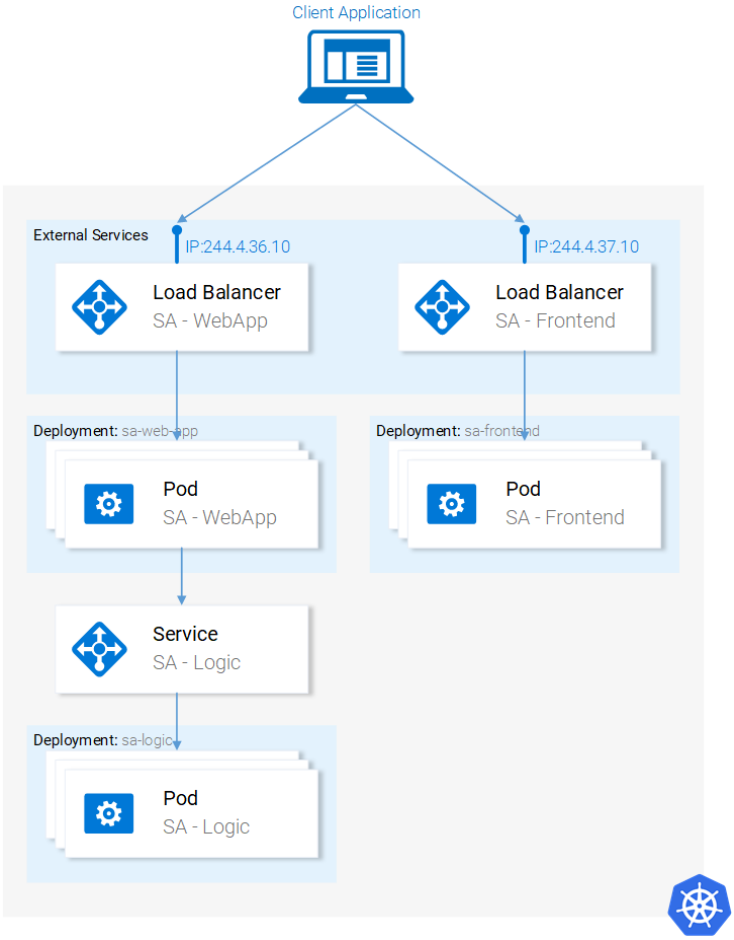

Рисунок 4.2 – Основні користувацькі сутності Kubernetes

Розглянемо послідовність розміщення мікросервісу в середовищі Kubernetes (Рисунок 4.3).

Основними елементами є створення Dockerfile, що необхідно для побудови контейнера. Після цього починається процесу розміщення контейнеру в середовищі Kubernetes. Спочатку необхідно створити k8s.yml файл, що містить в собі всі характеристики взаємодії сервісу та хмарного середовища. Після надання конфігураційного файла Kubernetes створює всі основні сутності. Deployments, replica sets, pods, services та інше. Після цього виконуються liveness check – спеціальні функції, які перевіряють роботу системи.

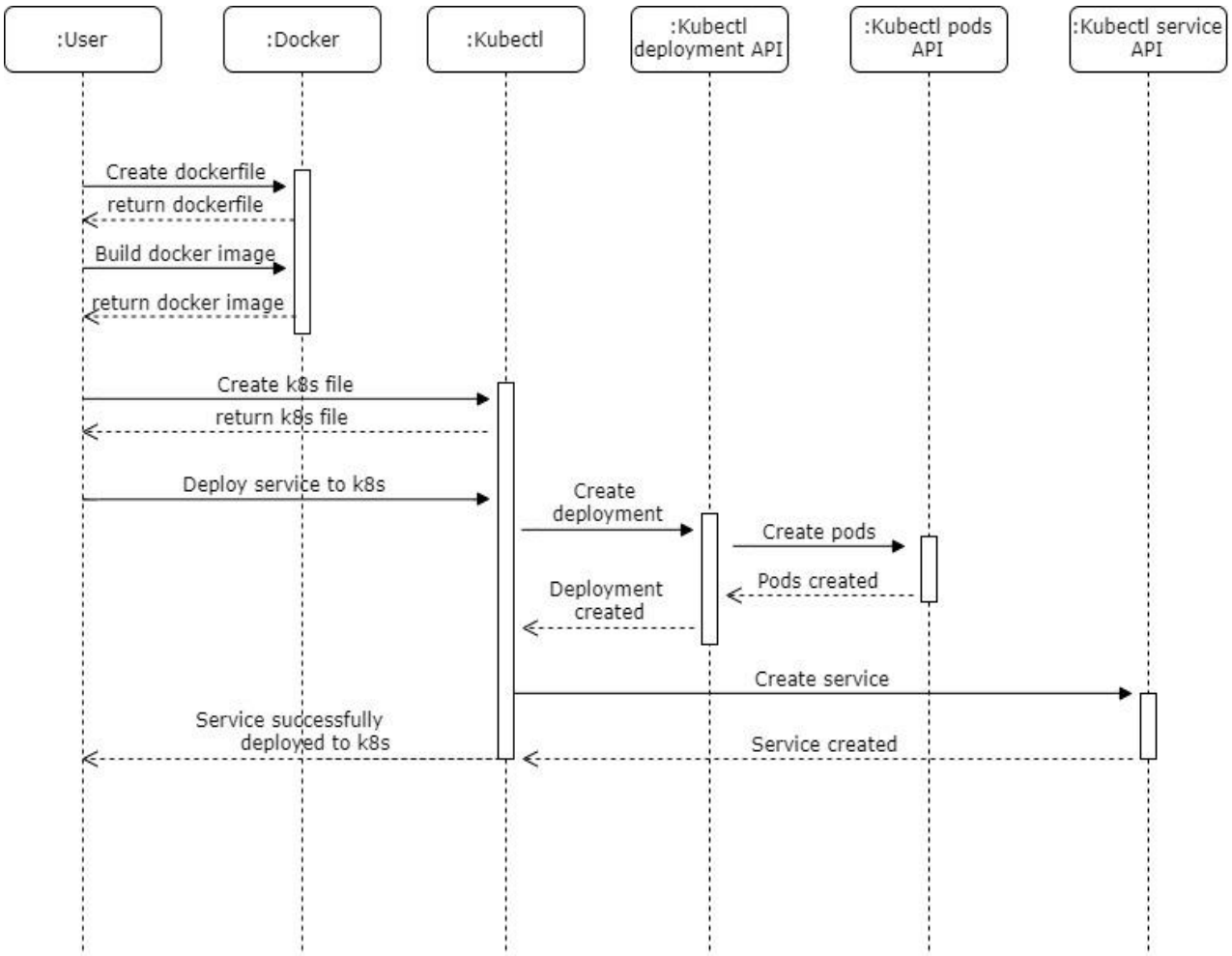

Рисунок 4.3 – Послідовність розміщення сервісу в інфраструктурі Kubernetes

Для того, щоб успішно розмістити додаток в хмарному середовищі, в першу чергу потрібно створити Dockerfile (Рисунок 4.4) – це спеціальний файл, який описує структуру docker контейнера, який буде створено. В цьому файлі має бути описана операційна система контейнера, та основні файли програмного додатку, які будуть переміщені в контейнер під час створення.

Після створення dockerfile за допомогою команди docker build нам потрібно створити docker image, який в подальшому буде розміщено в кластері Kubernetes. Docker зчитає всі конфігурації, прописані в Dockerfile і дасть вам доступ до новоствореного образу.

```
FROM buildkit-export AS buildkit-buildkitd.oci_only
COPY --from=buildkitd.oci_only /usr/bin/buildkitd.oci_only /usr/bin/
COPY --from=buildctl /usr/bin/buildctl /usr/bin/
ENTRYPOINT ["buildkitd.oci_only"]

# Copy together all binaries for containerd worker mode
FROM buildkit-export AS buildkit-buildkitd.containerd_only
COPY --from=runc /usr/bin/runc /usr/bin/
COPY --from=buildkitd.containerd_only /usr/bin/buildkitd.containerd_only /usr/bin/
COPY --from=buildctl /usr/bin/buildctl /usr/bin/
ENTRYPOINT ["buildkitd.containerd_only"]

FROM alpine AS containerd-runtime
COPY --from=runc /usr/bin/runc /usr/bin/
COPY --from=containerd /go/src/github.com/containerd/containerd/bin/containerd* /usr/bin/
COPY --from=containerd /go/src/github.com/containerd/containerd/bin/ctr /usr/bin/
VOLUME /var/lib/containerd
VOLUME /run/containerd
ENTRYPOINT ["containerd"]

FROM buildkit-${BUILDKIT_TARGET}
```

Рисунок 4.4 – Приклад dockerfile

Далі важливим є опис deployment для Kubernetes. Цю конфігурацію можна описати за допомогою k8s.yaml (Рисунок 4.5) файлу. В цьому файлі містяться основні дані про Docker образ, який треба розміщувати в середовищі, конфігурації додатків, кількість реплік, балансування навантаження та багато іншого.

Тип розміщення може відрізнятись: Deployment – розміщення декількох незалежних одне від одного реплік. Кожна репліка може бути вилучена або перезавантажена одна від іншої. StatefulSet – набір залежних реплік. Кожна репліка залежить від всіх інших і трафік не направляється на одну до тих пір, пока не будут запущені всі інші. Цей механізм може бути корисним для сервісів, які мають один розподілений стан між усіма мікросервісами, наприклад, якщо ми використовуємо репліковану базу данних, або спеціальних фреймворків для побудови розподілених кешів. Таким чином за допомою Kubernetes можна розміщувати в хмарному середовищі не тільки мікросервіси, але і інфраструктурні елементи, такі як бази даних, брокери повідомлень, key-value сховища даних, розподілені кеші, розподілені конфігуратори тощо.

```yaml
apiversion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  revisionHistoryLimit: 5
  minReadySeconds: 10
  selector:
      matchLabels:
        app: nginx
        deployer: distelli
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
    replicas: 3
    template:
      metadata:
        labels:
            app: nginx
            deployer: distelli
      spec:
        containers:
          - name: nginx
            image: nginx: 1.7.9
```

Рисунок 4.5 – Приклад k8s.yaml конфігурації

Після опису k8s.yaml файлу треба виконати основні команди kubectl інтерфейсу для застосування k8s.yaml файлу до кластера, або використати графічний інтерфейс (Рисунок 4.6) для розміщення додатку в кластері. Після цього запуститься автоматичний процес розміщення контейнера в середовищі, який створить всі необхідні залежності та забезпечить безшовне оновлення. Безшовне оновлення включає в себе виключення робочої репліки тільки після того, як нова версія сервісу буде повністю розміщена та доведена до робочого стану. Ця функціональність є дуже важливою для систем, де не допускається навіть хвилини простою для завантаження нової версії додатку.

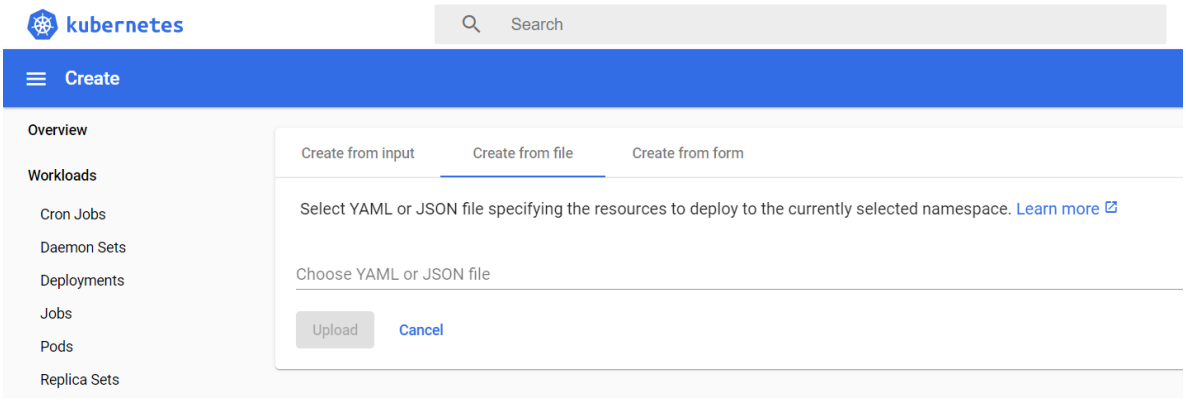

Рисунок 4.6 – Графічний інтерфейс для розміщення сервісу в кластері

Після цього Kubernetes створить в кластері сутність Deployment, яка виступає конфігуратором всього сервісу. Далі, на основі конфігурації з Deployment буде створено pods, інформація про які буде братися з розділу docker image в k8s файлі. Інформацію про створені pod-и можна отримати в графічному інтерфейсі Kubernetes (Рисунок 4.7).

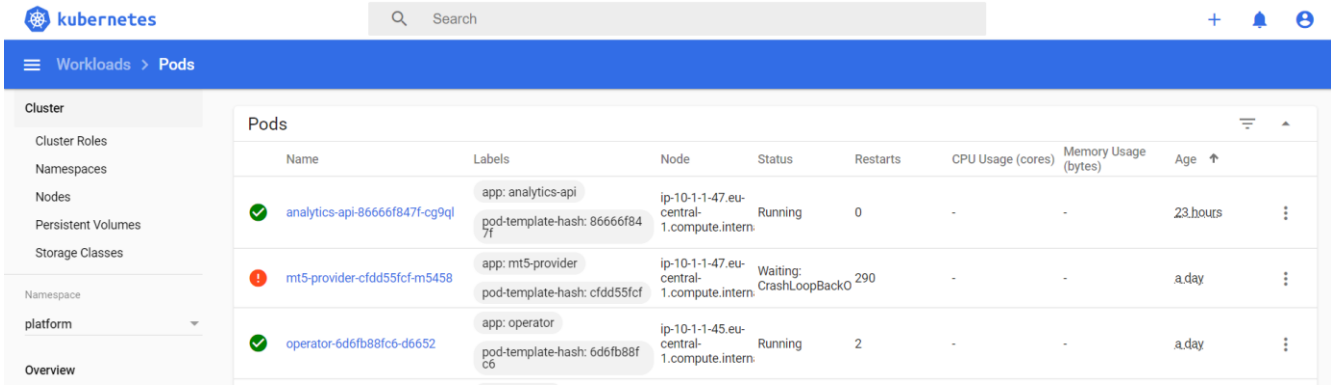

Рисунок 4.7 – Інформація про створені Pods в кластері

Після створення всіх необхідних ресурсів та сутностей Kubernetes, ви отримуєте повідомлення про готовність системи. Після цього мікросервіс стає повністю доступним з усією його функціональністю і включений в роботу системи.

4.2. Опис роботи системи для роботи з геоінформаційними даними

Розглянемо основні функції та можливості системи для обробки геоінформаційних даних.

Система для обробки геоінформаційних даних націлена на збереження даних вимірів пристроїв, створення зв’язків між пристроями та вимірами, прив’язка певних пристроїв до користувачів.

Одними із основних функцій системи є:

— Реєстрація користувачів та керування ними.

— Додавання нових пристроїв до системи.

— Створення полів та груп полів.

— Збереження вимірів пристроїв в систему.

— Робота з фільтрами: пошук вимірів по пристрою, по полям тощо.

— Відображення даних вимірів на карті.

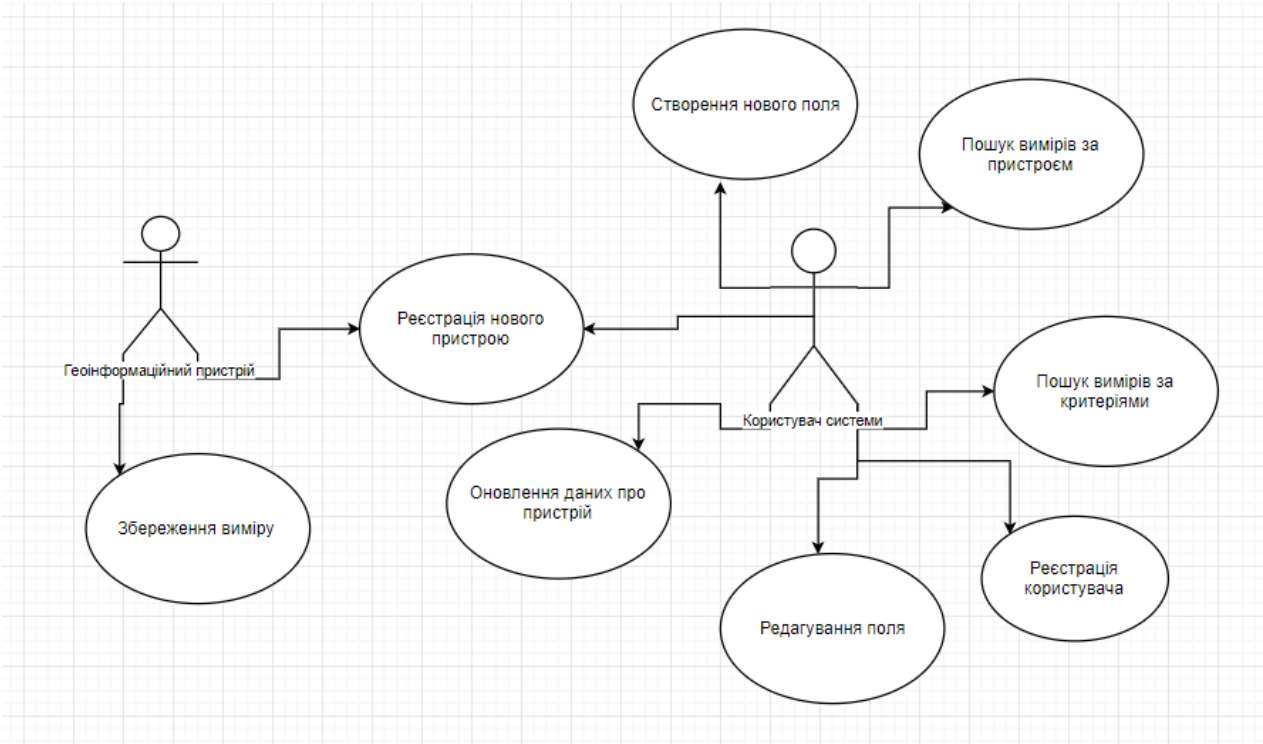

Рисунок 4.8 – Use-case діаграма основних можливостей системи

Розглянемо послідовність обробки окремих функцій системи.

Створення нового пристрою та робота з пристроями (Рисунок 4.9) в цілому. Пристрої дуже близько пов’язані з вимірами і дуже важливо мати можливість правильно сортувати дані в залежності від пристроїв.

Найважливішою функцією в цьому процесі є отримання всіх вимірів за пристроєм. Дуже часто користувач хоче знайти саме свої пристрої та отримати дані саме з них. Для реалізації цієї функції ми також робимо запит на шлюз, який відправляє запит на сервіс пристроїв. Сервіс пристроїв знаючи ідентифікатор пристрою робить запит на сервіс вимірів і таким чином сервіс вимірів отримує із БД про вимірі саме цього конкретного пристрою. Результат запиту повертається на сервіс вимірів, а з нього через сервіс пристроїв клієнту.

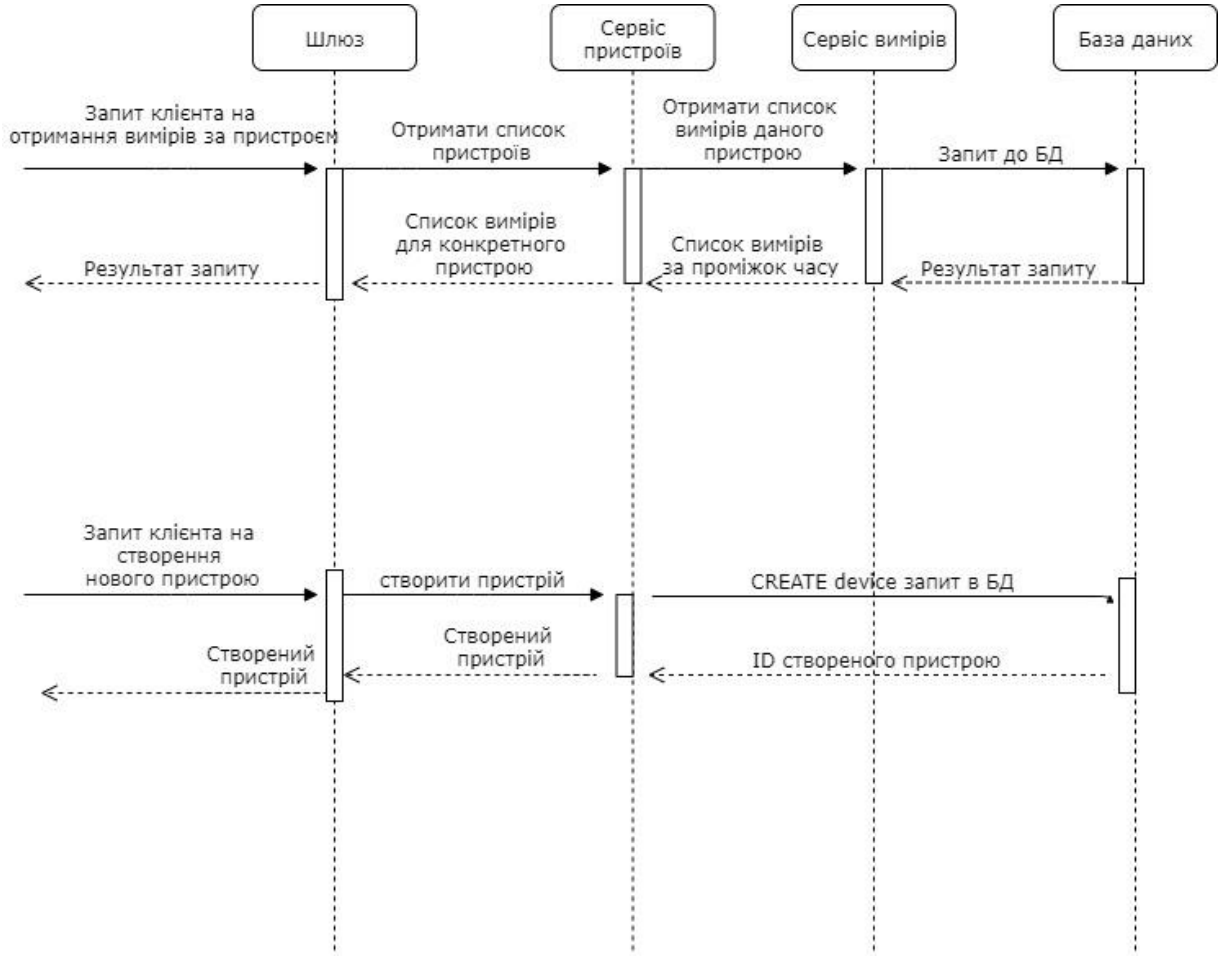

Рисунок 4.9 – Діаграма послідовності роботи з пристроями

Наступною функціональністю системи є робота з вимірами. Так як основою функціональністю системи є збір та обробка даних цих вимірів, то можна вважати цей процес найважливішим.

Всі виміри на сервер передаються виключно пристроями, створення яких описано вище. Пристрої можуть надсилати дані як за протоколом HTTP, так і з використання специфічних протоколів, таких як UDP, TCP тощо.

Послідовність реєстрації виміру в системі включає в себе запит на шлюз, який потім перенаправляє запит на сервіс вимірів. Сервіс вимірів в свою чергу робить запит на сервіс пристроїв, для верифікації інформації про реєстрацію даного пристрою в системі. Отримавши інформацію про пристрій сервіс вимірів робить запит у базу даних на створення нового запису про вимір. База даних повертає унікальний ідентифікатор створеного виміру.

Наступною операцією є отримання всіх вимірів за проміжок часу. Ця функціональність є важливою для побудови різноманітних графіків. Наприклад, для графіку залежності щільності грунту від часу виміру. Клієнт робить запит на шлюз, який перенаправляє запит на сервіс вимірів, в свою чергу клієнт робить запит у базу даних для отримання всіх записів за певний проміжок часу. Треба зауважити, що база даних має сконфігуровані b-tree індекси по полю дати створення виміру, для того, що пошук по цих полях був максимально ефективним.

Для правильного геопозиціонування на карті вимірів дуже важливим є функціонал створення на управління полями та їх групами. Поле – це координатна площина, яка може бути відображена на карті або іншому географічному зображенні.

Першим процесом в сервісі полів є створення поля (Рисунок 4.10). Створення поля проводиться користувачем за допомогою користувацького інтерфейсу. Користувач вводить в систему координати поля, або відмічає точки на карті, а вже з карти клієнтський додаток відправляє інформацію на шлюз. В свою чергу шлюз

знаходить сервіс вимірів і перенаправляє запит на створення. Сервіс формує INSERT запит в базу даних та створює новий запис про поле.

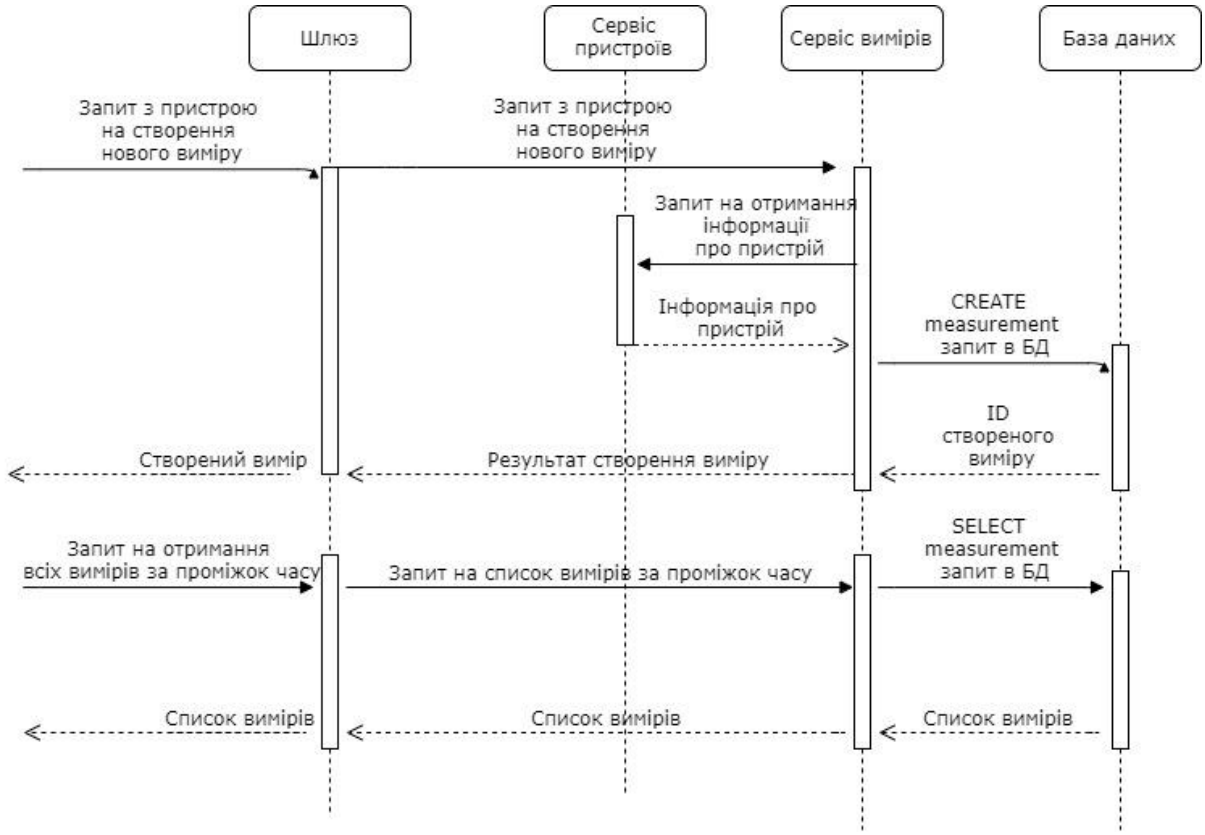

Рисунок 4.10 – Діаграма послідовності роботи з вимірами

Іншим елементом функціональності є формування запиту на реєстрацію виміру на певному полі (Рисунок 4.11). Так як відображення вимірів на певних областях, які іменуються полями є однією з ключових частин системи цей функціонал є дуже важливим і повинен бути правильно спроектованим.

При запиті з пристрою на шлюз про реєстрацію поля ми повинні визначити до якого саме поля належить даний пристрій. Як правило пристрої вміють передавати координати свого місцезнаходження, в іншому ж випадку це повинен виконати інженер.

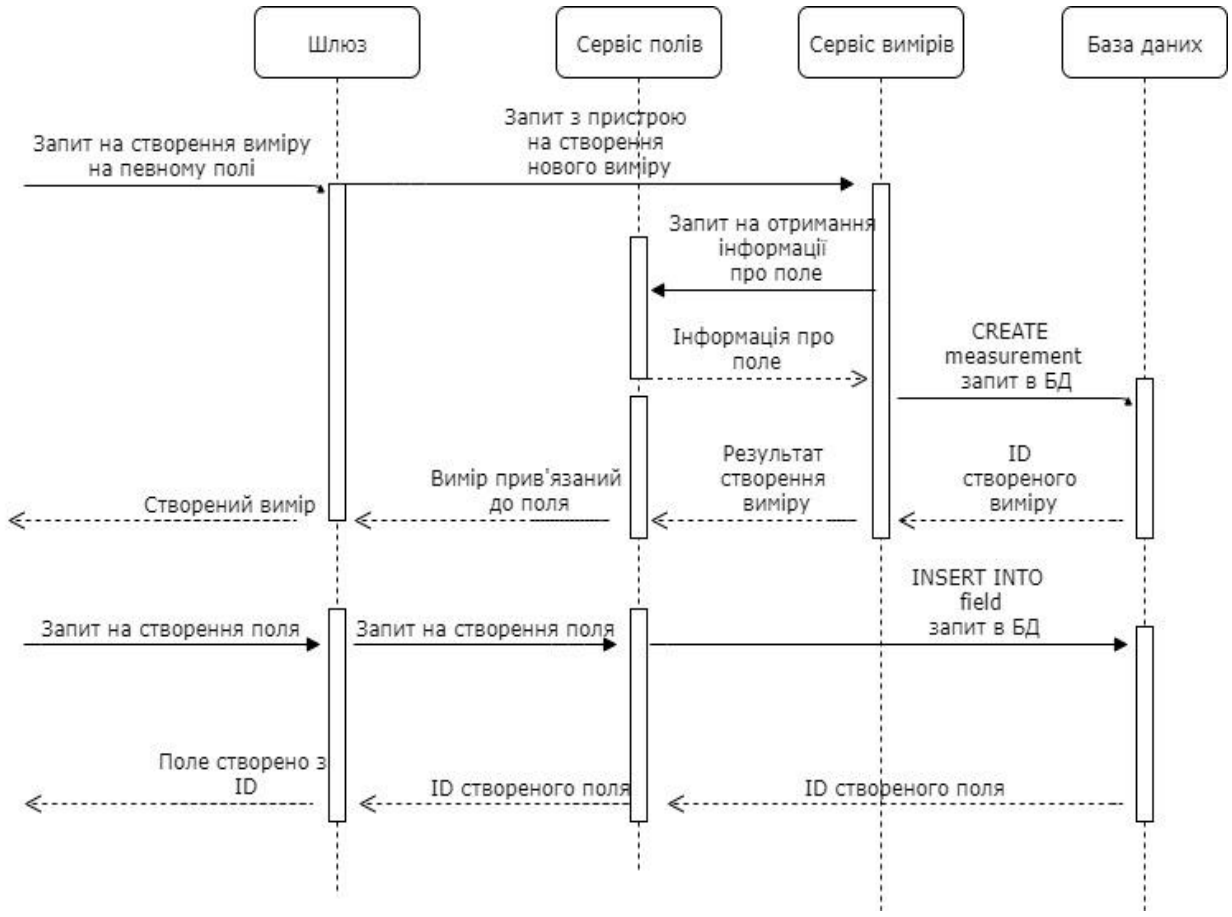

Рисунок 4.11 – Діаграма послідовності роботи з полями та вимірами

Таким чином процес виглядає так: запит зі шлюзу передається на сервіс вимірів на ендпоінт створення нового виміру на певному полі. Сервіс вимірів в свою чергу робить запит на сервіс полів для визначення ідентифікатора поля за координатами. Отримавши ідентифікатор поля сервіс дописує його в об’єкт виміру і формує запит в базу даних на реєстрацію виміру. INSERT запит відправляється в БД, яка в свою чергу генерує і повертає унікальний ідентифікатор виміру.

Цей ідентифікатор повертається через шлюз назад до клієнта (тобто пристрою) і таким чином пристрій отримує зворотній зв’язок про те, що його запит було успішно оброблено.

Отже розроблена система має велику кількість функцій, найважливіші з яких було описано вище. Всі модулі системи мають зв’язок один з одним, але завдяки

використанню мікросервісної архітектури було досягнуть максимально слабкої зв’язності. Всі сервіси комунікують між собою лише для отримання специфічних додаткових даних. Таким чином система стає набагато простішою в підтримці та супроводі. Також мікросервісний підхід дозволяє оновлювати всі модулі, такі як сервіс пристроїв, сервіс вимірів, сервіс полів окремо та незалежно один від одного без простою.

Висновки до розділу 4

Розроблений функціонал має велику кількість функцій, направлених, в першу чергу на збір та обробку даних з геоінформаційних пристроїв. Основними діючими елементами системи є користувач, який взаємодіє з системою за допомогою графічного інтерфейсу, та геоінформаційний пристрій, який можна зареєструвати в системі та передавати з нього зібрані дані в систему для подального аналізу та обробки.

Мікросервісна система розміщена в хмарному середовищі за допомогою фреймворку для оркестрації Kubernetes, який має всі необхідні частини для продуктивного використання хмарного середовища і ефективної розробки за допомогою мікросервісів.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З СИСТЕМОЮ

Запропонований програмний продукт виконаний у вигляді мікросервісів, які розміщуються в хмарному середовищі, за допомогою фреймворку для оркестрації Kubernetes. Мікросервіси виконані за всіма вимогами та згідно з завданням, реалізують в собі весь комплекс засобів та механізмів для ефективного виконання поставленої задачі по роботі з геоінформаційними даними.

Взаємодія з клієнтом виконана з дотриманням всіх основних принципів REST-архітектури. Основні переваги використання REST-API полягають в:

— Продуктивність — взаємодія компонентів є домінуючою властивістю сприйняття користувачами і результативності мережі;
— Простота єдиного інтерфейсу і авторизації;
— Можливість модифікації компонентів для задоволення мінливих потреб;
— Надійність — висока відмовостійкість при наявності збоїв у складі, роз'ємів або даних.

Для запуску системи необхідно виконати наступні кроки:

— Сконфігурувати серверний додаток за допомогою фреймворку Spring Boot;
— Побудувати Docker-контейнери створених сервісів;
— Створити конфігураційний файл для kubernetes k8s.yaml;
— Розмістити створені контейнери в системі Kubernetes.

5.1. Опис взаємодії з системою обробки геоінформаційних даних

Після запуску мікросервісів та розміщення користувацького графічного інтерфейсу користувач має доступ до таких функцій:

Авторизація користувача (Рисунок 5.1) у системі – відбувається за рахунок протоколу OAuth2, та потребує введення користувачем свого особистого логіну та паролю, виданого йому при додаванні в систему.

Рисунок 5.1 — Сторінка авторизації.

Користувач має можливість працювати та керувати полями в системі. (Рисунок 5.2). В правій частині екрану користувач може бачити графічне відображення поля на карті. В лівій частині екрану можна побачити список полів та спеціальній поля, за допомогою яких можна групувати поля, шукати поля по заданим критеріям, наприклад, за іменем. У кожного поля є колонка з групами, до яких належить поле та колонка з вимірами, які було зроблено на даному полі. При наведенні курсора на поле, це поле автоматично масштабується на карті, що дає користувачу змогу швидко зорієнтуватись з яким саме об’єктом на карті він працює.

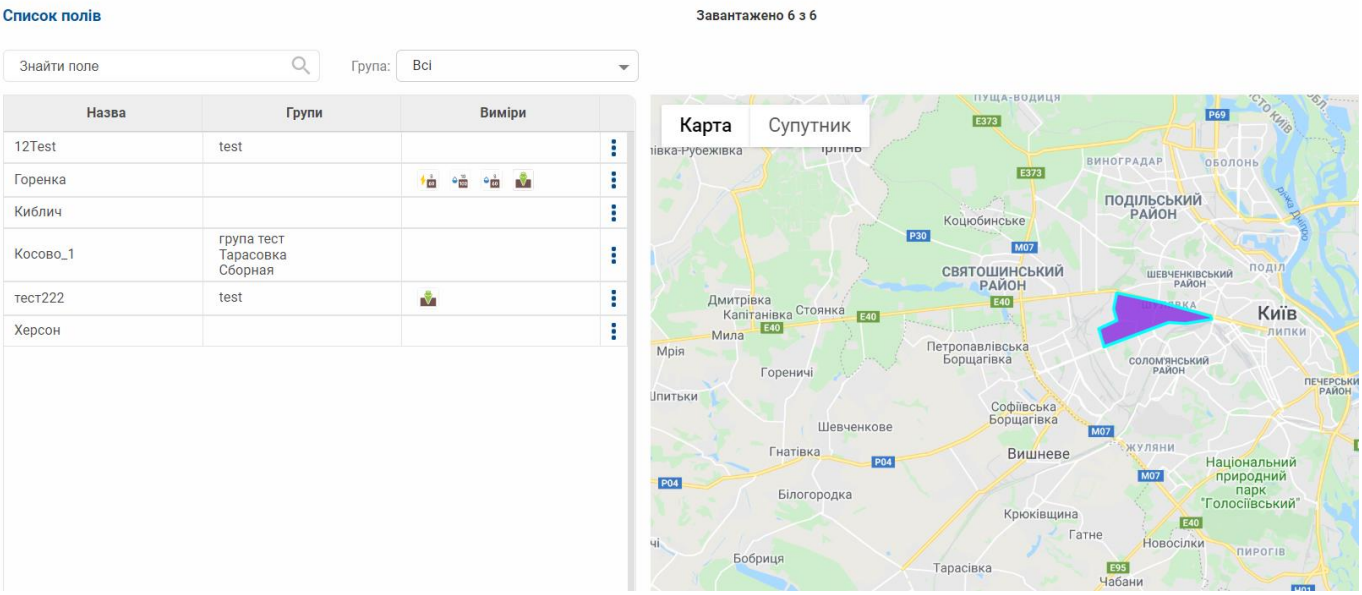

Рисунок 5.2 — Сторінка роботи з полями.

Користувач має можливість переглянути групи полів (Рисунок 5.3) до яких він має доступ. Групи полів створені для вирішення задачі групування полів за певними критеріями або логічними функціями. Наприклад, ви можете створити групу полів певного агрохолдингу для того, щоб в майбутньому фільтрувати всі дані саме с цієї групи, а не тільки з одного поля.

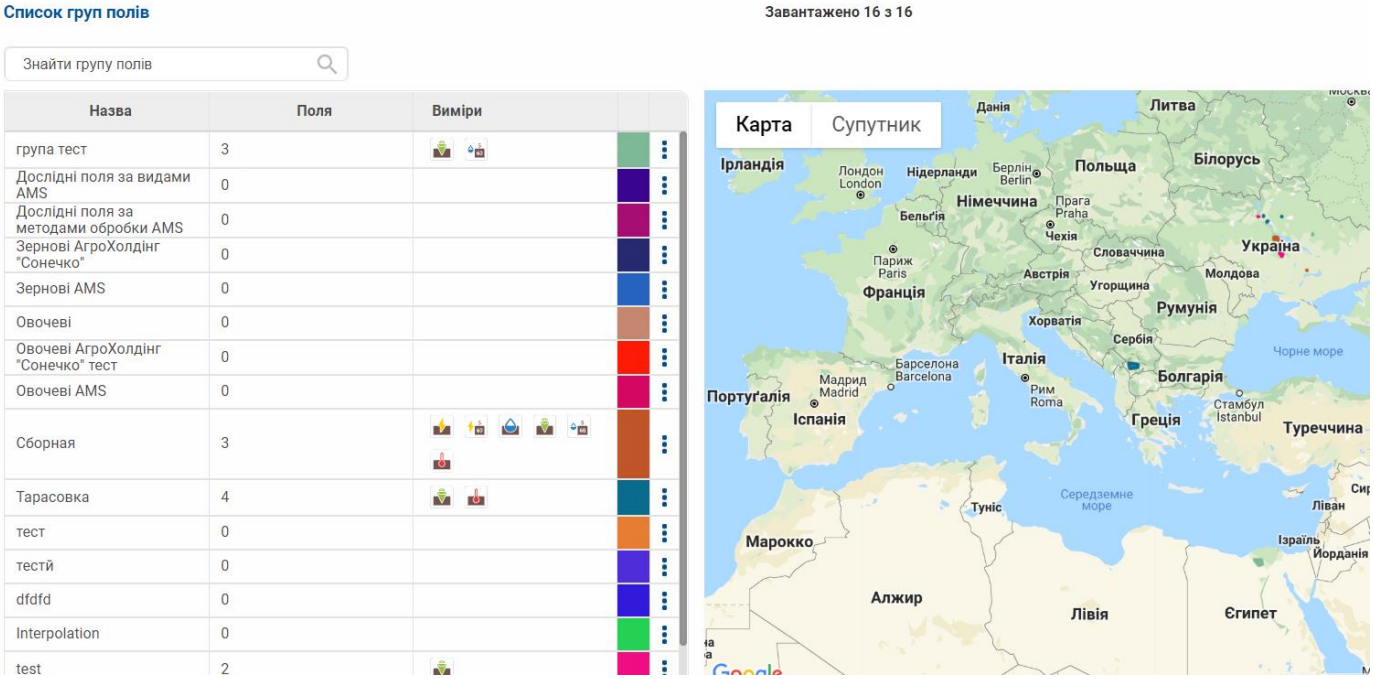

Рисунок 5.3 — Сторінка груп полів.

Для створення нового поля (Рисунок 5.4) користувач може використати макет карти і таким чином задати межі поля користуючись графічним інтерфейсом. Інший же спосіб – задати рамки поля за допомогою геометричних координат за допомогою спеціальної форми. Також поля можна попередньо зберігати в файл і завантажувати їх в систему.

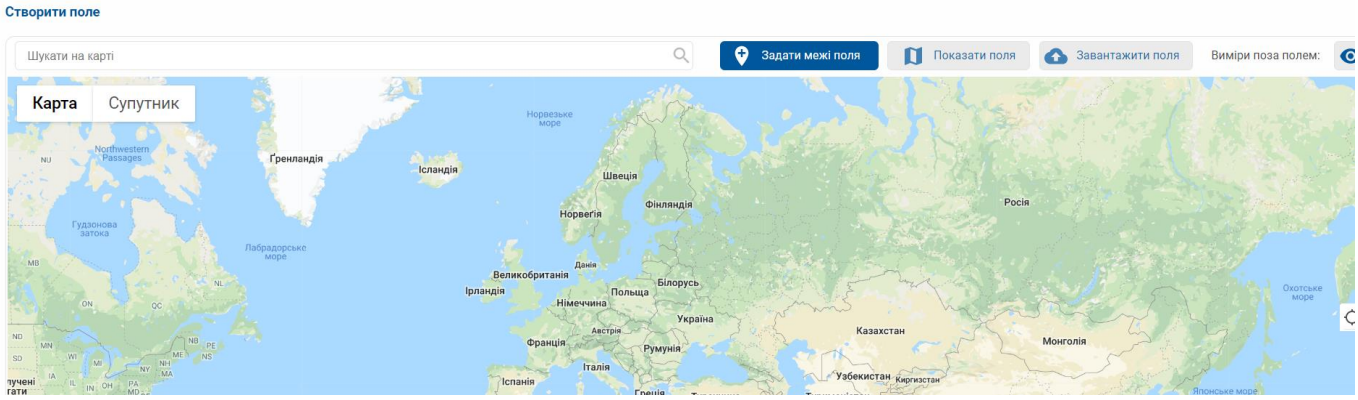

Рисунок 5.4 — Приклад відповіді сервера на запит про видалення.

На сторінці вимірів користувач може отримати інформацію про ущільнення грунту на всіх ділянках за певний проміжок часу, зібраних з одного пристрою (Рисунок 5.5). Стандартна одиниця виміру – кПа (кілопаскалі), також користувач може обрати альтернативні одиниці вимірів, наприклад, кгс/см^2 (кілограм-сила на сантиметр в квадраті). Одиниця виміру щільності грунту знаходиться на осі ординат. По осі абсцис знаходиться глибина, на якій було зроблено даний вимір.

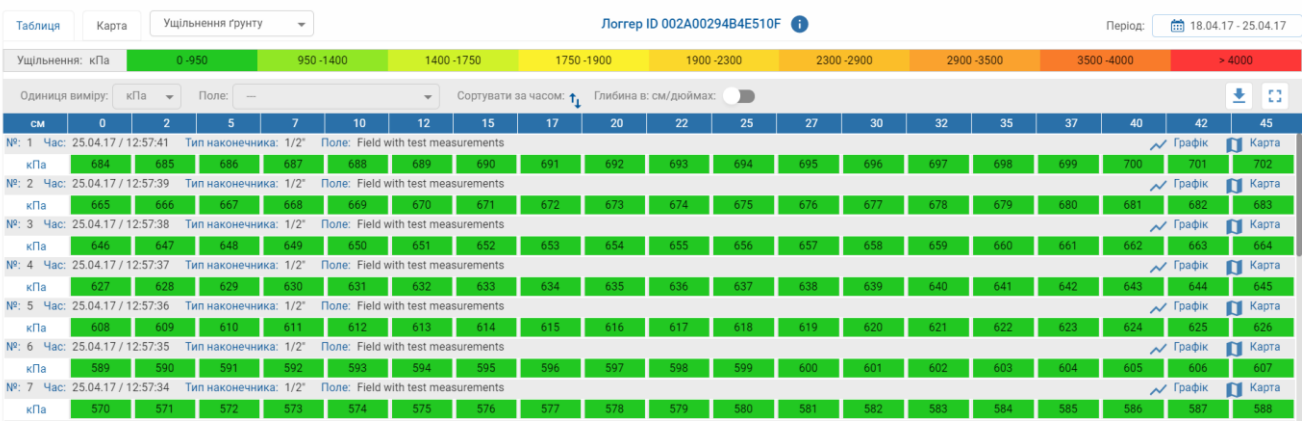

Рисунок 5.5 — Виміри за пристроєм.

На іншій сторінці користувач може отримати інформацію про зібрані виміри (Рисунок 5.6) на певному полі (географічно розмежованій ділянці). Основні елементи сторінки еквівалентні сторінці з вимірами по пристрою, але інформація сортується та фільтрується саме за полем, незалежно від пристроїв, з яких було зібрано інформацію.

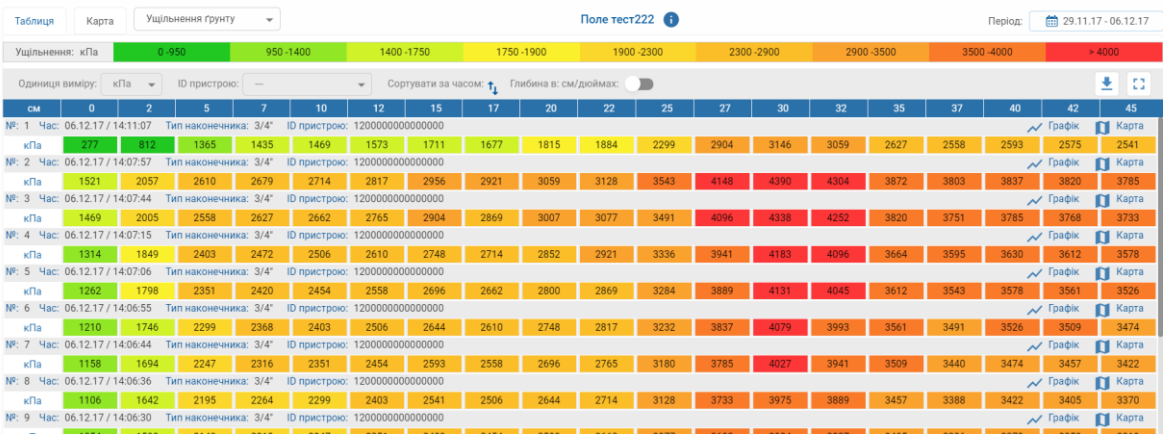

Рисунок 5.6 — Виміри за полем

Однією із функцій на сторінках з вимірами є побудова графіків залежності ущільнення грунту (Рисунок 5.7) від глибина, на яку було занурено пенетроментр. Цей графік будується по групі вимірів за певний проміжок часу на певному полі або для певного пристрою. Також є можливість друку цього графіку.

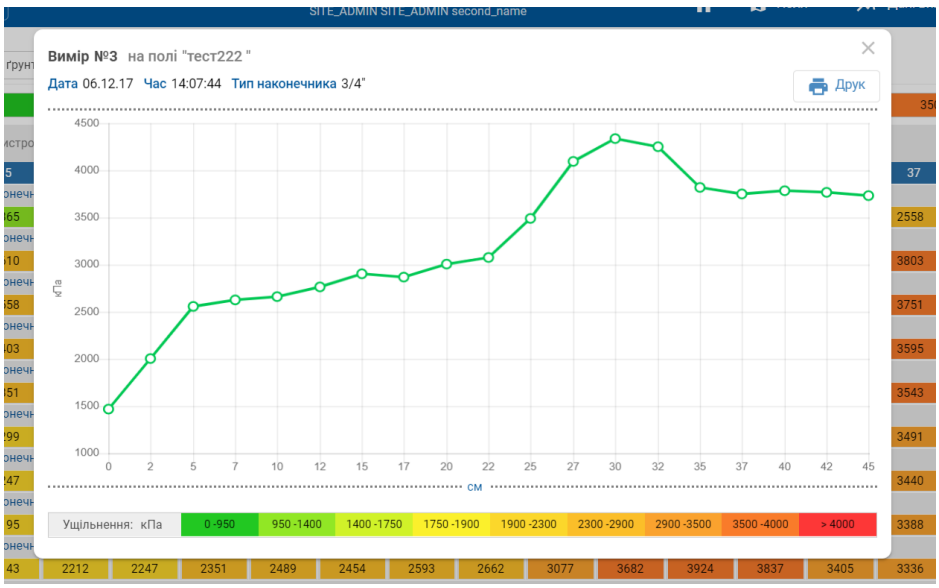

Рисунок 5.7 — Графік вимірів на полі

Наступною функцією системи є відображення певного виміру на карті (Рисунок 5.8). На карті відображається попередньо створене поле, на якому проводився вимір, координати точки, де було зроблено конкретний вимір, дату виміру та час. Також відображається інформація про технічні характеристики пристрою, таки як тип накінечника.

Функція відображення вимірів на карті або іншому географічному зображенні є дуже важливою для аналізу отриманих із пристроїв вимірів та формування повної наукової картини в геоінформаційній інженерії.

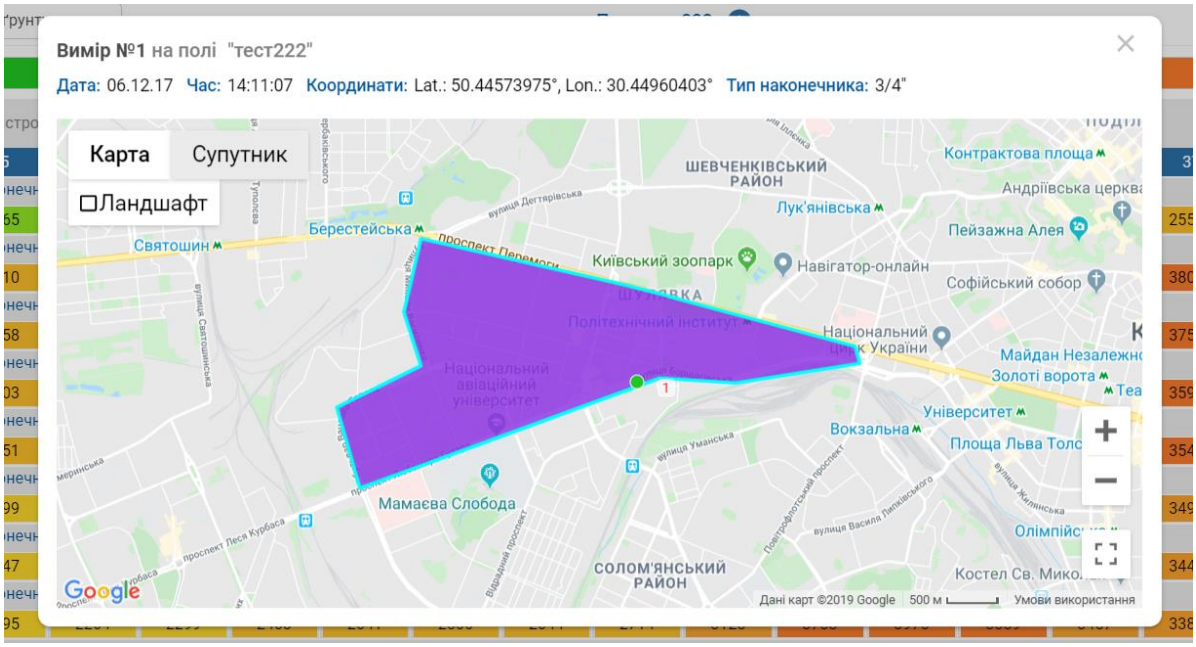

Рисунок 5.8 — Відображення вимірів на полі.

Висновки до розділу 5

Отже, розроблений користувацький графічний інтерфейс має багато функцій на користувацькому інтерфейсі, які дозволяють правильно проаналізувати дані, отримані з пристроїв, конфігурувати різні настройки під різних користувачів. Дозволяють працювати з різними пристроями в системі. А також формувати поля на картографічній площині для більш інформативного відображення інформації.

6. БІЗНЕС-ПЛАН ІННОВАЦІЙНОГО ПРОЕКТУ

Ідея проекту полягає у реалізації мікросервісу обробки геоінформаційних даних в хмарному середовищі. Переваги технології хмарних обчислень включають високу продуктивність, економію в витратах, високу степінь доступності та легку масштабованість.

Проте, при її практичній реалізації виникає цілий ряд ще не вирішених наукових проблем, які перешкоджають повноцінному використанню всіх потенційних переваг такого підходу.

Одна з них полягає в тому, що для забезпечення можливості практичного використання хмарного середовища в різних областях необхідно організувати універсальну систему для запуску індивідуальних додатків.

Розв’язком вказаної проблеми є використання мови програмування Java, як мови програмування для реалізації додатку для доступу та керування хмарним середовищем, забезпечення безпечного доступу користувачів ресурсоємних додатків в розподіленому хмарному середовищі.

6.1 Опис ідеї стартап-проекту

Зараз на ринку програмних застосунків є досить велика кількість прикладних програм, що можуть бути використані у навчальних закладах для процесу керування проектами. Вони відрізняються між собою інструментарієм та призначенням. Проте всі вони не можуть бути інтегровані в існуючу систему моніторингу і виконувати свої функції.

Проаналізувавши деякі з таких систем можна зробити висновок, що дані рішення або вузьконаправлені за сприятливою ціною, або підходять для широкого спектру завдань і в той же час занадто дорогі, коли система потребує не один, а групу модулів. Зважаючи на це, є потреба в використанні більш зручної, інтуїтивно

зрозумілої системи для кафедри, яка може бути використана як самостійне рішення та модуль для вже існуючих програмних рішень. Результати аналізу представлені у таблиці 6.1.

Таблиця 6.1. Основні ідеї в проекті

Сутність ідеї	Можливості застосування	Переваги користувача
Створення системи призначеної для задачі обробки геоінформаційних даних в хмарному середовищі	IT компанії (розробники, адміністратори) — розробники високонавантажених додатків, які потребують великих обчислювальних потужностей; — фахівці у сфері інформаційних технологій, які будуть адмініструвати цю систему	Компонент повинен забезпечити зручний програмний інтерфейс для керування та взаємодії з хмарними сервісами. Також особливістю компонента є можливість керування конкретними характеристиками та специфікаціями хмарного сервера, що дозволяє користувачам легко конфігурувати середовище в залежності від своїх задач.

У подібних систем на ринку є аналоги, відрізняються вони тим, що не вирішують конкретно поставлену задачу, мають відмінну сферу застосування та не працюють з потрібною кількістю параметрів. Як правило додатки використовують анлійську мову, на багато дорожчі та старіші. А розроблена мікросервісна система була створена в першу чергу з розрахунком на цільову аудиторію українського ринку.

Тому доцільно проводити аналіз потенційних техніко-економічних переваг ідеї порівняно з пропозиціями конкурентів. Результат аналізу у таблиці 6.2.

Таблиця 6.2. Визначення характеристик ідеї проекту

Техніко-економічні характеристики ідеї	Продукція конкурентів			Слабкі (W), нейтральні (N) та сильні (S) сторони		
Назва продукту	Cloud Stack	Eucalyptus	VCloud			
Операційна система та версії	Платформи Windows 10, 8.1, 7	Платформи Windows 10, 8.1	Платформи Windows 7 (SP1)			
Системні вимоги	1.4 gigahertz (GHz);	2.5 gigahertz (GHz);	1.4 gigahertz (GHz);			
Розміри	4 Gb	8 Gb	4 Gb			
Мови програмування	Java	C++	Java			

Як результат в дослідженні характеристик (слабких та нейтральних) ідеї проекту виконано аналіз технічних та аналітичних характеристик конкурентів, які вже існують на ринку, а це надає можливість спрогнозувати частку майбутніх потенційних користувачів. Тож, запропоноване ПЗ вирізняється конкурентноспроможним потенціалом серед перелічених вище характеристик, адже містить близько 5% в якості показників слабкої сторони, 45% - нейтральної сторони, 60% - сильної сторони. А, враховуючи, що основна мета майбутнього ПЗ –

удосконалення вже існуючих систем з подібним функціоналом, маємо досить успішні наміри завоювати прихильність на ринку.

6.2 Технологічний аудит ідеї проекту

Для проведення технічного аудиту ідеї проекту, потрібно провести аудит технології, за допомогою якої можна реалізувати ідею проекту. І для початку потрібно визначити можливість технологічної здійсненності проекту. Результат представлений у таблиці 6.3.

Таблиця 6.3. Можливість технічної реалізації проекту

№ п/п	Суть проекту	Основні використані технології	Доступність технологій
1.	Надання чіткого зрозумілого інтерфейсу для керування та розміщення додатків у хмарному середовищі	HTML, CSS, JavaScript, TypeScript, Angular	Доступні для вільного користування
2.	Прийом та обробка запитів	Java, Spring boot, http, tomcat	Доступні для вільного користування
3.	Можливість керування конкретними характеристиками та специфікаціями хмарного сервера	Openstack java API	Доступні для вільного користування
4.	Можливість створення нових віртуальних машин	OpenStack cloud service, Openstack java api	Доступні для вільного користування

5.	Механізм аутентифікації та авторизації користувача.	Мова програмування Java, метод шифрування/дешифрування даних за допомогою закритого ключа RSA	Доступні для вільного користування

Обрана технологія реалізації ідеї проекту: програмний продукт планується розробити у середовищі Jet brains Intellij Idea, використовуючи об’єктно-орієнтовану мову програмування Java, бібліотека для роботи з файлами “Files”, метод шифрування/дешифрування даних RSA.

6.3 Аналіз можливостей для запуску стартап-проекту на ринку

Аналіз можливостей ринку, використаних для правильного впровадження на ринку, загроз, які виникають на ринках, які можуть стати перешкодою для реалізації визначеного проекту, з його допомогою можна визначити основні напрями для розвитку визначеного проекту та урахувати стан в ринковому середовищі, потреби для можливих клієнтів, пропозицій компанії-конкурентів. В першу чергу для цього необхідно проаналізувати попит (таблиця 6.4).

Таблиця 6.4. Аналіз характеристик для потенційного для стартап-проекту ринку

№ п/п	Критерії стану ринку (найменування)	Основна характеристика
1.	Число основних учасників, од	2 учасники
2.	Об’єм продажів, грн/ум.од	$3 млрд
3.	Характеристика динаміки ринку	зростає
4.	Основні обмеження для входу	ліцензійні умови для інтегрування, забезпечення маркетиногової стратегії для розробки грамотної рекламної кампанії

5.	Окремі вимоги щодо стандартів та сертифікатів	відсутні
6.	Усереднена рентабельність у сфері, %	+15%

На основі розглянутих характеристик для показників стану ринку ІТ (в контексті ідеї стартап-проекту), можна зробити висновок, що ринок є привабливим для входження за попереднім оцінюванням.

Та враховуючи, справді, велику необхідність у розробці подібних програмних комплексів, задля покращення процесів створення програмних продуктів та полегшення роботи розробникам, доцільно, необхідно і має сенс створення даної автоматизованої системи. І виявляється необхідним входження на ринок з такою розробкою.

Аналіз можливостей ринків, які використовуються для впровадження проекту на ринку, загроз ринку, потенційно перешкоджаючих проекту, надає змогу планувати розвиток проекту та враховувати стан середовища на ринку, основні потреби клієнтів.

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи (таблиця 6.5).

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають.

Результати представлені у таблицях 6.6 та 6.7 відповідно.

Таблиця 6.5. Аналіз можливих користувачів описаного стартап-проекту

№ п/п	Ринкова потреба	Опис цільового сегменту ринку	Різні потенційні цільові групи	Потенційні вимоги груп користувачів

1.	Створення системи для запуску ресурсоємних додатків в розподіленому середовищі	IT компанії (розробники, адміністратори)	IT компанії (розробники, адміністратори) — розробники високонавантажених додатків, які потребують великих обчислювальних потужностей; — фахівці у сфері інформаційних технологій, які будуть адмініструвати цю систему	Компонент повинен забезпечити зручний програмний інтерфейс для керування та взаємодії з хмарними сервісами. Також особливістю компонента є можливість керування конкретними характеристиками та специфікаціями хмарного сервера.

Аналіз характеристик клієнтів показав, що цільовою аудиторією є IT компанії. Для всіх аудиторій важлива цілодобова технічна підтримка, якість продукції та технічні характеристики.

Таблиця 6.6. Основні загрожуючі фактори

№ п/п	Фактор	Опис характеру загрози	Потенційна відповідна реакція
1.	Недосвідчені учасники команди	Призначення недосвідчених працівників (студентів) для виконання роботи проекту ставить під загрозу дату його завершення, оскільки їм може знадобитися більше часу, щоб ознайомитися з бізнес-моделлю, технологіями.	Щоб мінімізувати цей ризик, необхідно закласти достатньо часу на введення нових працівників у курс справи.
2.	Планування та послідовність виконання задач	Навіть якщо ці завдання виконують різні люди, їх одночасне виконання у великій кількості створює ризик для проекту, особливо наприкінці його реалізації.	Перевірити, чи не заплановано забагато завдань на один і той самий час. При плануванні задач проекту спочатку необхідно скласти список завдань і згрупувати їх, щоб оцінити весь обсяг проекту та кінцеві результати. Потім можна починати зв’язувати завдання, щоб отримати ідеальний розклад.

Таблиця 6.6 (продовження)

3.	Потужна клієнтська база конкурентів	Конкуренти, які мають впевнений досвід продукта на ринку здобули сильну базу клієнтів-споживачів.	Розвиток вражаючої маркетингової кампанії, створення стратегії піар-менеджменту, закладання регламенту рекламної кампанії, акційних пропозиції.

Таблиця 6.7. Фактори можливостей

№ п/п	Фактор	Зміст	Можлива реакція компанії
1.	Аналіз досвіду конкурентів	Формування стратегії реалізації проекту без навчання на своїх помилках, а при навчанні на помилках конкурентів – невдалі рекламні, маркетингові ходи конкурентів.	Планування і реалізація проекту з максимальним виключенням ймовірності виникнення помилок вже досвідчених конкурентів на ринку споживачів.
2.	Націлення продукту на основні функціональності, які відстутні у конкурентів	Реалізація нових можливостей для споживачів, впровадження покращень суміжного з	Чітке планування задач, розподілення задач між розробниками з залученням прорахованих ризиків,

		конкурентами проекту функціональностями.	мотивація команди ідеєю кінцевого продуту.

Таблиця 6.7 (продовження)

3.	Підвищення рентабельності проекту	За рахунок правильного планування всіх етапів проекту, чіткого формулювання бізнес-моделі є можливість залучення до команди проекту студентів в якості розробників.	Зниження кількості інвестицій для розробки і впровадження кінцевого продукту.

Надалі проводиться аналіз пропозиції – визначаються загальновизначені елементи конкуренції (таблиця 6.8): аналізуються типи можливої майбутньої конкуренції, конкурентоспроможність та рівень боротьби під час конкуренції, товарними типами та ознаками галузі.

Таблиця 6.8. Характеристика конкуренції на ринку за ступенями

Тип середовища для конкуренції	Ознаки проявлення характеристики	Як чинить вплив на підприємство
1. Олігополістична конкуренція	В більшій мірі конкурентна, але лідерів можна явно виділити	Важко вийти на міжнародний рівень
2. Глобальний рівень конкурентної боротьби	Конкурентнами можуть бути різні країни світу	Розвиток на українській IT арені та вихід на ринок

Таблиця 6.8 (продовження)

3. Конкуренція в середині галузі	Спостерігається в пропозиціях на покупку ПЗ, якості функціональностей.	Розробка вузьконаправленого програмного забезпечення
4. Конкуренція в типах товарів: - товарно-типова	Конкуренція між програмними забезпеченнями одного типу	Випуск кращих і якісніших версій ПЗ, взаємодія з пропозиціями і побажаннями споживача.
5. За певними перевагами	Функціональні можливості програмного забезпечення	Розширити функціональні можливості
6. За характеристикої інтенсивності на ринку	Для споживачів має значення «бренд»	Створення добре відомої марки

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі (таблиця 6.9) - за моделлю п’яти сил М. Портера, яка вирізняє п’ять основних факторів, що впливають на привабливість вибору ринку з огляду на характер конкуренції:

- конкурент, що вже є у галузі;
- потенційні конкуренти;
- наявність товарів-замінників;
- постачальники, що конкурують за ринкову владу;
- споживачі, які конкурують за ринкову владу.

Врахувавши аналіз конкуренції, який було проведено у таблиці 6.9, а також враховуючи характеристики проекту (таблиця 6.2), посилаючись на вимоги споживачів до товару (таблиця 6.5) та фактори середовища (таблиці 6.6 і 6.6) було визначено та обгрунтовано фактори конкурентноспроможності. Аналіз було оформлено за таблицею 6.9, фактори було обгрунтовано за таблицею 6.10.

Таблиця 6.9. Г за М. Портером

	Прямі галузеві конкуренти	Можливі конкуренти	Основні постачальники	Користувачі	Потенційні замінники товару
Складові аналізу	1. CloudStack 2. Eucalyptus 3. OpenStack	Наявність товарних знаків, доступ до ресурсів	Основним постачальником є інтернет-ресурси	Торгівельні знаки, система інформації	відсутні
Висновки:	Конкурентна боротьба неінтенсивна так, як прямі конкуренти більше спеціалізуються на інших функціональних можливостях	Є можливості входу на ринок за рахунок гнучкості цін; потенціальна конкуренція є серед існуючих компаній	Зазвичай постачальники не диктують умови співпраці	Умови клієнтів в залежності від ситуації постійно змінюються	-

Висновки: аналіз конкуренції в галузі за М. Портером показав, що можлива робота на арені ІТ України так, як конкурентна боротьба не інтенсивна і прямі конкуренти більше спеціалізуються на інших функціональних можливостях, також проект повинен відповідати умовам споживачів, які в залежності від ситуації можуть змінюватись.

Таблиця 6.10. Визначення основних факторів для конкурентоспроможності

№ п/п	Основний фактор	Опис фактора
1.	Споживчі потреби	Споживчі потреби задають необхідність розробки проекту
2.	Результат	Завжди досягається кінцевий результат
3.	Потенціал на ринку	Використання не за призначенням
4.	Цінові та собівартосні характеристики	Не переоцінена, достатньо конкурентна ціна
5.	Технічне обслуговування	Випуск нових версій продукту

За результатом обгрунтування видно, як споживчий фактор потреб таких, як проаналізоване формування мінімальних поверхонь, дослідження різних природніх факторів, а також ціни.

За даними конкурентними факторами було проведено характеристуку різних сторін проекту, який наведено у таблиці 6.11.

Таблиця 6.11. Характеристика сильних та слабких якостей проекту

№ п/п	Конкурентні фактори	Бали 1-20	Рейтинг програм-конкурентів у порівнянні з системою «AWS»						
			-3	-2	-1	0	+1	+2	+3
1.	Споживчі потреби	12				+			

2.	Основна результативність	15					+		
4.	Ціностворення продукції	10			+				
5.	Обслуговування	15						+	

Таблиця 6.12. Аналіз проект за схемою SWOT

Сильні сторони: Цілодобова технічна підтримка; справка-інструкція по експлуатації; якість продукту; продукт відповідає потребам споживачів; доступність.	**Слабкі сторони**: Низька репутація компанії на початку впровадження проекту в життя; присутність багів.
Можливості: Вихід на міжнародний ринок; результативність; розвиток нових функціональних можливостей.	**Загрози**: Зниження доходів потенційних клієнтів; блокування реклами на просторах інтеренету, соцільних мереж; блокування інтернет-ресурсу програмного забезпечення.

Порівняльний аналіз сильних та слабких сторін «мікросервісу обробки геоінформаційних даних в хмарному середовищі» показав, що результативність і маркетинговий потенціал, а також ціна та собівартість продукції є сильними факторами конкурентоспроможності у порівняні з найближчим конкурентом - пакетом «CloudStack».

SWOT- аналіз стартап-проекту вказав на сильні сторони, якими є цілодобова підтримка, інструкція по експлуатації, якість продукту, відповідність потребам

споживачів та доступність. А слабкими сторонами є низька репутація компанії на початку впровадження проекту в життя та присутність багів.

На основі SWOT-аналізу розробляються альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок.

Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів (таблиця 6.13).

Таблиця 6.13. Альтернативи ринкового впровадження стартап-проекту

Альтернатива ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
Вихід на нові ринки	Пошук інвесторів	1-6 місяців
Розширення виробничої лінії	Пошук інвесторів	Після виходу на ринок основного продукту, до 6 місяців

Отже, спочатку потрібно вивести на основний ринок розроблену систему, а вже потім шукати можливості розширення програмного функціоналу для користувачів.

6.4 Розроблення ринкової стратегії

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів, які визначені у таблиці 6.14.

Таблиця 6.14. Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту) за рік	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1.	Навчальні заклади	Готові	25000	Велика конкуренція	Середня

В якості цільової груп потенційних споживачів було обрано навчальні заклади. Інтенсивність конкуренції в сегменті невелика в Україні та вхід у сегмент є легким, через високий попит на внутрішній ІТ арені. За результатами аналізу потенційних груп споживачів (сегментів) автори ідеї обирають цільові групи, для яких вони пропонуватимуть свій товар, та визначають стратегію охоплення ринку.

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку, яка визначається у таблиці 6.15.

Таблиця 6.15. Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоп-лення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1.	Проведення конференції для	Ексклюзивний розподіл	Відповідна ціна, довіра до бренду.	Стратегія лідерства по

	закордонних користувачів			витратах

Для обраної альтернативи розвитку проекту було обрано ексклюзивний розподіл, а стратегію лідерства по витратах, як базову стратегію розвитку. Тому що, така стратегія передбачає, що компанія за рахунок чинників може забезпечити більшу, ніж у конкурентів маржу між собівартістю товару і середньоринковою ціною. Вибір стратегії конкурентної поведінки визначається у таблиці 6.16.

Таблиця 6.16. Визначення базової конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурен-та, і які?	Стратегія конкурентної поведінки
1.	Проект не є першопрохідцем	Компанія буде забирати існуючих споживачів у конкурентів і шукати нових	Основні характеристики товару будуть схожими	Стратегія позиціонування

При визначенні базової стратегії конкурентної поведінки до даного проекту, який не є першопрохідцем, було обрано стратегію позиціонування. Компанія показує чим відрізняється продукт від конкурентів, чим корисний, які є переваги над конкурентами, таким чином відбувається позиціонування на особливостях, які важливі споживачу.

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту, а також в залежності від обраної базової стратегії розвитку

та стратегії конкурентної поведінки розробляється стратегія позиціонування (таблиця 6.16), що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку чи проект.

Таблиця 6.16. Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкуренто-спроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1.	Ціна, якість	Знизити ціни на продукцію та створити якісний товар	Відповідна ціна, довіра до бренду	Створення сайтів, надійність, якість

При визначенні стратегії позиціонування були обрані вимоги до товару цільової аудиторії такі, як ціна та якість. Обрано базову стратегію розвитку – знизити ціни на продукцію та створити якісний товар; асоціації було обра-но на базі вимог цільової аудиторії, які формують комплексну позицію про-екту- сторення сайтів, надійність, якість.

6.5 Розроблення маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у таблиці 6.18 підсумовані результати попереднього аналізу конкурентоспроможності товару.

Таблиця 6.18. Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує ПЗ	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1.	Технічна підтримка	Своєчасна технічна підтримка	Відповідна ціна, довіра до бренду
2.	Адаптований інтерфейс користувача	Асоціативне використання новими користувачами	Наявність «Справки»

При визначенні ключових переваг концепції потенційного товару було обрано вигоду, яку пропонує товар – своєчасну технічну підтримку, ключовими перевагами перед конкурентами є відповідна ціна та довіра до бренду, також ще одна вигода – це адаптований інтерфейс користувача, ключовими перевагами якого є наявність «Справки» по експлуатації продукту. Надалі розроблена трирівнева маркетингова модель товару: уточняються ідея продукту, його фізичні складові, особливості процесу його надання (таблиця 6.19).

Таблиця 6.19. Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="3">Створити систему для запуску ресурсоємних додатків в розподіленому середовищі</td></tr>
<tr><td rowspan="4">II. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Надання чіткого зрозумілого інтерфейсу для керування та розміщення додатків у хмарному середовищі
2. Прийом та обробка запитів
3. Можливість керування конкретними характеристиками та специфікаціями хмарного сервера
4. Можливість створення нових віртуальних машин
5. Можливість захисту від втрати даних</td><td>-</td><td>-</td></tr>
<tr><td colspan="3">Якість: стандарти, нормативи, параметри тестування</td></tr>
<tr><td colspan="3">Марка: «OpenStack manager»</td></tr>
<tr><td>III. Товар із підкріпленням</td><td colspan="3">До продажу: потребує ознайомлення з роботою системи
Після продажу: підтримка клієнтів</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: патенту та комерційної таємниці</td></tr>
</table>

За задумом проект забезпечує можливість реалізації системи, що використовується у сфері розробки ресурсоємних додатків в розподіленому середовищі. До продажу клієнти мають ознайомитися з роботою проекту, а після

продажу буде цілодобова технічна підтримка. За рахунок патенту та комерційної таємниці товар буде захищено від копіювання.

Визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар, яке передбачає аналіз ціни на товари-аналоги, а також аналіз рівня доходів цільової групи споживачів описано в таблиці 6.20.

Таблиця 6.20. Визначення меж встановлення ціни

Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни
50 – 52000 $	500 – 5000 $	30 – 50 $

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (таблиця 6.21): проводити збут власними силами чи залучати сторонніх посередників, вибір та обґрунтування оптимальної глибини каналу збуту, вибір та обґрунтування виду посередників.

Таблиця 6.21. Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Клієнти купують продукт безпосередньо у компанії-розробника	дослідницька робота зі збору маркетингової інформації	Нульовий рівень: тільки виробник	Через сайт виробника

При зазначеній специфіці закупівельної поведінки цільових клієнтів, що клієнти купують продукт безпосередньо у компанії - розробника, було обрано оптимальну систему збуту - через сайт виробника так, як це найпростіший спосіб придбання ПЗ для цільових клієнтів.

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (таблиця 6.22).

Таблиця 6.22. Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідолення	Концепція рекламного звернення
1.	Клієнти дізнаються про нові продукти з реклами в інтернеті, соціальних мереж, по рекомендаціям інших людей	Інтернет, соціальні мережі	Відповідна ціна, довіра до бренду.	- Поширення знань про продукт - Інформація про випробування товару	- Перелік основних правдивих даних про продукт - Науково-професійний стиль

Проаналізувавши специфіку поведінки цільових клієнтів, було обрано концепцію рекламного звернення:

1. Перелік основних правдивих даних про продукт;
2. Науково-професійний стиль.

Реклама буде поширюватись через інтернет та соціальні мережі. Завданням рекламного повідомлення є зацікавлення та поширення знань про продукт новим клієнтам, та поширення інформації про випробування товару.

Висновки до розділу 6

Отже, ринкова (маркетингова) програма орієнтовано має бути побудована таким чином:

– створення продукту;

– пошук потенційних клієнтів;

– базова стратегія розвитку – стратегія диференціації, тобто конкурентоспроможність формується шляхом надання споживачеві бажаного товару. На основі ретельного вивчення споживчого середовища розробляється одна чи декілька відмітних характеристик власного товару;

– стратегія конкурентної поведінки – стратегія виклику лідера, тобто на споживчому ринку орієнтуватись на всіх можливих споживачів, у тому числі клієнтів фірм-конкурентів. Така стратегія передбачає принцип «идти по пятам» за лідером ринку. За подальші цілі ставиться можливість обігнати лідерів цільового сегменту.

Стан та динаміка ринкового середовища на сьогоднішній день і ще багато років є і будуть залишатись сприятливими для впровадження розробленої системи, а також для її необхідності.

Конкурентні переваги створеного продукту очевидні. На вітчизняному ринку аналогів, а відповідно і конкурентів не виявлено. У той час, як попит на програмні системи подібного роду тільки набирає популярність. Самостійних систем, які

створені для моделювання процесів кардіореспіраторної системи людини під впливом гіпоксії, в Україні (цільовому ринку) немає. Хоча схожих програмних розробок достатньо за кордоном, на інших мовах, проте існуючі аналоги мають у рази меншу кількість параметрів, що розраховуються та потребують наявності АПК для свого функціонування, що дуже сильно збільшує ціну. На міжнародному ринку конкуренція з'явиться та постійно буде рости, якщо не підтримувати та не розвивати свій продукт.

Також, після проведення аналізів можливого цільового сегменту (споживачів), потреб споживачів та можливого попиту, динаміки ринку та рентабельності роботи на ринку, можна однозначно зробити висновок, що створений проект доцільний до комерціалізації.

Перспективи впровадження з огляду на потенційні групи клієнтів, бар'єри входження, стан конкуренції та конкурентоспроможності проекту – прямі, і тільки доводять можливість впровадження, та не марну розробку створеного продукту.

Отже, є достатньо обґрунтовані передумови, що дозволяють зробити припущення про те, що реалізоване програмне забезпечення «Мікросервіс обробки геоінформаційних даних в хмарному середовищі» може бути успішно застосовано для різних цільових груп, в тому числі, для навчальних закладів. В якості детальної аргументації надано розгорнутий аналіз запланованого проекту у вигляді маркетингового аналізу стартап-проекту, план організації стартап-проекту, фінансово-економічного аналізу та оцінки ризиків проекту, планування заходів з комерціалізації проекту.

ВИСНОВКИ

Результатом даної роботи є прототип програмного інтерфейсу. Він задовольняє більшості висунутих вимог, може легко розширяться за рахунок особливості внутрішньої структури (без зміни внутрішньої логіки). Згодом, грунтуючись на цьому прототипі можна реалізувати повноцінний програмний інтерфейс до хмарного сервісу.

Програмна система має клієнт-серверну архітектуру. Для реалізації сервісного шару системи для керування хмарним середовищем, використана технологія Spring MVC, для реалізації зв’язку між додатком та хмарним середовищем OpenStack було використано бібліотеку з відкритим кодом apache jcloud.

В ході цієї роботи мною була проведена розробка прототипу для інтеграції з хмарної середовищем, створений функціонал для надання можливості користувачеві керувати хмарним середовищем, там використвувати його для розміщення високонавантажених додатків. Ці розробки дозволять користувачам проводити повномасштабні експерименти над великим об'ємом даних, надають наукову цінність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Amar Kapadia Implementing Cloud Storage with OpenStack Swift/ Amar Kapadia, Sreedhar Varma, Kris Rajana. — L.:Packt Publishing, 2014. — 140p.
2. Амосов А. А. Вычислительные методы для инженеров / А. А. Амосов, Ю. А. Дубинский, Н. П. Копченова. — М: Мир, 1998. – 644 с.
3. Бахвалов Н. С. Численные методы / Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков. — Москва: Бином, 2001. — 363 с.
4. Боровков А. И. Компьютерный инжиниринг / А. И. Боровков. — СПб: Изд-во Политехн. ун-та, 2012. — 93 с.
5. Портнякин И. Эффективные пользовательские интерфейсы / Иван Портнякин. — Санкт-Петербург: Лори, 2011. — 600 с.
6. Маркелов А. OpenStack. Практическое знакомство с облачной операционной системой / Кей Хорстманн. — М: ДМК Пресс, 2016. — 160с.
7. Менеджмент проекту Agile [Електронний ресурс]. — Режим доступу: https://agile.management.org/article/Agile.
8. Сазерленд Д. Scrum: Революционный метод управления проектами / Д. Сазерленд. — М. : Мир, 1982. — 114 с.
9. Royce W. Managing the Development of Large Software Systems / W. Royce. — Cambridge University Press, 6th ed., 1970. — 677 p.
10. Басс Л. Архитектура программного обеспечения на практике / Л.Басс, П.Клементс, Р.Кацман. — Сп. : Питер. — 2006. — 576 с.
11. Арлоу Д. UML и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, А. Нейштадт. — Сп. : Символ-Плюс. — 2007. — 624 с.
12. Грофф Д. SQL: полное руководство / Д. Грофф, П. Вайнберг. — М. : Наука, 2005. — 608 с.

Додаток 1

Мікросервіс обробки геоінформаційних даних в хмарному середовищі

Апробація

УКР.НТУУ“КПІ”.ТР4270_19М

Аркушів 4

2019