Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Юрій КРАВЧЕНКО
«_____»_____2022 року

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технологій»

на тему:

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ОЦІНКИ ВАРТОСТІ РОЗГОРТАННЯ ДОДАТКУ У ХМАРНОМУ СЕРЕДОВИЩІ

Виконав: студент групи МІТм -21

Денис ТИМОШЕНКОВ _____
(ім'я та ПРІЗВИЩЕ) (підпис)

Керівник: доцент кафедри мережевих та інтернет технологій
(посада)

к.т.н. Оксана ГЕРАСИМЕНКО _____
(науковий ступень, вчене звання, ім'я та ПРІЗВИЩЕ) (підпис)

Київ 2022

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Юрій КРАВЧЕНКО
«_____»_____2022 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувачу вищої освіти Тимошенкову Денису Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи:
Інформаційна технологія оцінки вартості розгортання додатку у хмарному середовищі
затверджена на засіданні кафедри МІТ «31» серпня 2022 р. протокол №1
2. Термін здачі закінченої роботи «05» грудня 2022 р.
3. Вихідні дані до проекту (роботи)
Документація хмарних середовищ (AWS, Azure, Alibaba, GCP)
4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 70-80 стор.)
Вступ
1. Дослідження методів та засобів оцінки вартості розгортання додатку у хмарному середовищі. Постановка задачі
1.1 Огляд і аналіз існуючих моделей хмарних обчислень
1.2 Проблеми оцінки вартості розгортання додатку у хмарному середовищі
1.3 Постановка задачі
2. Розробка інформаційної технології оцінки вартості розгортання програмного додатку у хмарному середовищі
2.1. Вибір підходів та засобів для реалізації інформаційної технології
2.2. Концептуальна модель оцінки вартості розгортання програмного додатку у хмарному середовищі
2.3. Реалізація програмного забезпечення для оцінки вартості розгортання програмного додатку у хмарному середовищі
3. Розробка додатку агрегатора сервісів
3.1. Вибір засобів реалізації додатку
3.2. Проектування додатку агрегатора сервісів
3.3. Реалізація додатку агрегатора сервісів
3.4. Тестування агрегатора сервісів
Висновки

5. Перелік графічного матеріалу 8-10 слайдів

Дата видачі завдання 01.09.2022р.

Керівник роботи _____ Оксана ГЕРАСИМЕНКО
(підпис) (посада, ім'я, ПРІЗВИЩЕ)

Завдання прийняв до виконання _____ Денис ТИМОШЕНКОВ
(підпис) (ім'я, ПРІЗВИЩЕ)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	24.10.2022	
2	Розділ 1	01.11.2022	
3	Розділ 2	15.11.2022	
4	Розділ 3	01.12.2022	
5	Доповідь та слайди	05.12.2022	
6	Пояснювальна записка	05.12.2022	

Здобувач вищої освіти _____ Денис ТИМОШЕНКОВ
(підпис)

Керівник _____ Оксана ГЕРАСИМЕНКО
(підпис)

РЕФЕРАТ

Пояснювальна записка: 78 с., 17 рис., 6 табл., 4 додатки, 30 джерел.

Об’єкт дослідження: процес вибору хмарного провайдера та сервісів для розгортання додатку у хмарному середовищі.

Предмет дослідження: процес оцінки вартості розгортання додатку.

Мета роботи (проекту): створення засобу, здатного на основі інформації про тарифи від різних провайдерів хмарних ресурсів та вимог користувача надати оцінку вартості різних варіантів розгортання програмного додатку.

Методи дослідження: системний підхід, метод порівняння, низхідне проектування.

У спеціальній частині проведено опис та аналіз існуючих моделей хмарних обчислень, розглянуто «інстанс» як одиницю виконання обчислювальної роботи, досліджено їх типи та характеристики.

У роботі проведено аналіз існуючих моделей ціноутворення різних провайдерів хмарних ресурсів, досліджено проблеми оцінки вартості розгортання додатку у хмарі.

Запропоновано реалізацію агрегатора сервісів, призначеного для оцінки вартості різних варіантів розгортання програмного додатку на основі інформації про тарифи від провайдерів хмарних ресурсів та вимог користувача.

Розроблено вебдодаток для оцінки вартості розгортання додатку у хмарному середовищі.

Практичне значення роботи полягає у наданні розробникам програмного забезпечення нового інструменту для спрощення процесу вибору потенційного постачальника хмарних послуг.

Результати здійснених у кваліфікаційній роботі досліджень можуть бути використані підприємствами або розробниками ІТ для зменшення витрат на розгортання додатків у хмарному середовищі.

Наукова новизна роботи полягає у розробці простого у користуванні та невибагливого до ресурсів агрегатора сервісів для вибору найменшого по вартості варіанту розгортання вебдодатку без знань моделей ціноутворення та тарифів провайдера хмарних послуг.

Напрямки подальших досліджень включають розширення послуг агрегатора, додавання нових типів сервісів, послуг провайдерів хмарних ресурсів.

Ключові слова:

ХМАРНІ ОБЧИСЛЕННЯ, РОЗГОРТАННЯ ДОДАТКУ, ПРИВАТНА ХМАРА, ПУБЛІЧНА ХМАРА, ГІБРИДНА ХМАРА, АГРЕГАТОР СЕРВІСІВ, ПРОВАЙДЕР ХМАРНИХ ПОСЛУГ.

ЗМІСТ

Стор.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІТ – інформаційні технології.

ІС – інформаційна система.

БД- база даних.

СУБД – система управління базами даних.

РБД – реляційна база даних.

РСУБД – реляційна система управління базами даних.

DML – Data Manipulation Language (мова маніпулювання даними).

DDL – Data Definition Language (мова опису даних).

ООБД – об’єктно-орієнтована база даних.

ОРСУБД – об’єктно-реляційна СУБД.

ОРБД – об’єктно-реляційна база даних.

ОС – операційна система.

ЦП – центральний процесор.

ОЗП – оперативний запам’ятовуючий пристрій.

IaaS – Інфраструктура як послуга

PaaS – Платформа як послуга

SaaS – Програмне забезпечення як послуга

ВСТУП

Хмарні технології відіграють важливу роль у сфері інформаційних технологій та стають все більш популярним як серед організацій, так і серед розробників. Розгортання програмного забезпечення у хмарному середовищі надає цілий ряд переваг, включаючи масштабованість, економію коштів та доступ до додатків і послуг через Інтернет. Деякі хмарн є приватними, тобто використовуються лише в межах однієї організації або середовища, інші є загальнодоступними або гібридними. Масштабованість є однією з найбільших переваг хмарних технологій. Не маючи потреби у фізичному обладнанні, організації можуть швидко і легко збільшувати свої обчислювальні потужності в міру необхідності. Це дозволяє організаціям масштабувати свою інфраструктуру для обслуговування більшої кількості користувачів або швидкого запуску нових додатків без значних попередніх витрат. Економія коштів є ще однією перевагою хмарних технологій. Використовуючи хмарні обчислення, організації можуть заощадити гроші на фізичному обладнанні, використовуючи замість нього віртуальні машини. Це дозволяє легко тестувати нове програмне забезпечення або швидко розгортати середовища. Крім того, розгортання у хмарному середовищі може допомогти зменшити витрати, пов'язані з управлінням та обслуговуванням апаратного забезпечення.

Актуальність досліджень. Існує багато факторів, які впливають на вартість послуг хмарних ресурсів: це витрати на різні ресурси, такі як обчислювальна техніка, сховища даних, мережеве та програмне забезпечення, а також такі як плата за передачу даних та розташування в певному регіоні або зоні доступності.

Процес оцінки вартості розгортання хмарних технологій складається з декількох етапів:

1. першим кроком є визначення ресурсів, які будуть потрібні для підтримки додатків і робочих навантажень у хмарі. Це включає визначення кількості

та типу необхідних обчислювальних, сховищних та мережевих ресурсів, а також будь-якого спеціалізованого програмного забезпечення або послуг, які будуть використовуватися;

2. далі необхідно визначити, як ці ресурси будуть використовуватися з часом, включаючи обсяг і тип даних, які будуть зберігатися і оброблятися, а також очікувані моделі робочого навантаження;
3. оцінка витрат. Після отримання чіткого уявлення про необхідні ресурси і моделі використання, можна переходити до оцінки витрат на розгортання додатків у хмарі. Це може включати використання інструментів оцінки витрат або консультації з хмарними провайдерами для отримання детальних цінових політик.

Однією з ключових, але ще не охоплених проблем оцінки вартості розгортання, є величезна різноманітність доступних послуг провайдерів хмарних технологій і моделей ціноутворення. Кожен провайдер пропонує унікальний набір послуг і моделей ціноутворення, що ускладнює порівняння витрат і визначення найбільш економічно вигідного варіанту. Крім того, моделі ціноутворення, що використовуються хмарними провайдерами, можуть бути складними і важкими для розуміння, з широким спектром змінних і факторів, які можуть вплинути на кінцеву вартість. А, отже, на вибір провайдеру сервісу та типу послуги буде витрачено багато ресурсів.

Ступінь наукової розробки. Тема хмарних середовищ є достатньо популярною у наукових роботах, існує багато літератури та ресурсів щодо моделей та підходів до оцінки вартості, провайдери пропонують спеціальні калькулятори, які допомагають розробникам в оцінці. Проте тема порівняння та вибору послуг серед багатьох провайдерів ще не мала достатнього розвитку. У даній роботі представлено агрегатор сервісів провайдерів хмарних ресурсів, який на основі критеріїв, заданих користувачем, надає найкращі пропозиції за ціною з усіх комбінацій моделей ціноутворення, типів послуг, сервісів провайдерів хмарних технологій

Практичне значення одержаних результатів. Розроблена інформаційна технологія допоже організаціям та розробникам у процесі вибору послуг

провайдерів хмарних технологій та зменшить витрати на розгортання програмного забезпечення.

1. ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ ОЦІНКИ ВАРТОСТІ РОЗГОРТАННЯ ДОДАТКУ У ХМАРНОМУ СЕРЕДОВИЩІ. ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд і аналіз існуючих моделей хмарних обчислень

Розгортання програмного забезпечення – це процес розгортання, конфігурації та оновлення програмних додатків для забезпечення максимальної оптимізації, безпеки та сумісності в IT-середовищі.

Таке розгортання може відбуватися вручну або автоматично, в той час, коли працівники організації вважають його найменш критичним для роботи системи. Для великих розгортань це може призвести до поетапних випусків, які допомагають мінімізувати перебої в роботі співробітників та зменшити навантаження на систему, включаючи надання підтримки після розгортання.

Регулярне розгортання програмного забезпечення може відігравати важливу роль у зростанні та безпеці підприємства, особливо коли воно є своєчасним та послідовним. Оновлення і виправлення можуть допомогти задовольнити і відреагувати на бізнес-потреби організації, що постійно змінюються. Це і є цикл випуску програмного забезпечення.

Оскільки кількість витоків даних продовжує зростати в усьому світі, IT повинні відігравати ще більш активну роль, роблячи все можливе, щоб випереджати злочинних хакерів, багато з яких добре організовані, розумні і старанні у своєму прагненні знайти і використати будь-яку слабкість, що існує в об'єкті атаки. Регулярне стороннє тестування на проникнення, інтелектуальна система оповіщення та подальше вдосконалення програмного забезпечення через певні проміжки часу допомагають підтримувати безпеку та критично важливі цілі при розгортанні програмного забезпечення.

Також жодна програма не є безпомилковою або непроникною для зовнішнього втручання. І чим довше певна версія залишається розгорнутою без оновлення, тим більша ймовірність того, що вразливості в системі безпеки стануть причиною зовнішньої атаки. Часто ці вразливості відомі і для них вже існують готові патчі, але програмні виправлення залишаються нерозгорнутими. Це може статися з різних причин, але найчастіше це пов'язано з браком ресурсів в IT-відділі. Оптимізація робочих процесів затвердження для розгортання програмного забезпечення може допомогти пом'якшити ці проблеми. Підприємства можуть розглянути можливість прискореного відстеження цих розгортань – прийняття більш своєчасного мислення DevOps.

Розгортання програмного забезпечення може дати більше, ніж підвищення безпеки, оскільки додавання нових функцій, які відповідають конкретним потребам бізнесу, може допомогти підвищити продуктивність співробітників і впорядкувати робочі процеси.

Зі зростанням популярності та розвитку веб-додатків, веб-сервіси постійно розвиваються, щоб задовольнити запити клієнтів незалежно від їх географічного розташування. Це, по суті, дає бізнесу можливість досягти глобальної присутності, розширюючи свою аудиторію з мінімальними витратами (наприклад, на придбання та управління фізичною обчислювальною інфраструктурою, центрами обробки даних, серверами, заходами безпеки тощо). Це призвело до появи нової парадигми – хмарних обчислень.

Хмарні обчислення – це надання IT-ресурсів/обчислювальних послуг через Інтернет (тобто "хмару"). Від обчислювальної потужності та зберігання даних до розгортання програмного забезпечення та аналітики, ці послуги замінюють традиційний спосіб володіння фізичними активами, пропонуючи інновації, конкурентні переваги, гнучкість та спільне використання ресурсів. Фундаментальна концепція хмарних обчислень полягає в тому, щоб дозволити користувачам скористатися всіма їх перевагами, не будучи експертами в кожній технології, що лежить в їх основі. Однак, такі деталі, як розташування сервісу та варіанти масштабування, все ще залишаються важливими компонентами в рамках хмарного

процесу, які можуть бути змінені залежно від оптимальних умов роботи програми та вимог користувача.

Згідно з дослідженням, проведеним компанією Synergy Research Group., загальний дохід світових хмарних сервісів у 2022 році склав 217 мільярдів доларів США (рис. 1.1), при тому, що за дослідженням Gartner Inc., загальна ринкова вартість світових хмарних сервісів у 2019 році складала 227,8 мільярда доларів США. Для багатьох компаній є очевидним, що впровадження хмарних технологій як частини їхньої бізнес-стратегії залишатиметься головним пріоритетом в осяжному майбутньому. Одними з найбільших хмарних провайдерів є Microsoft Azure, Amazon Web Services (AWS) та Google Cloud Platform (GCP).

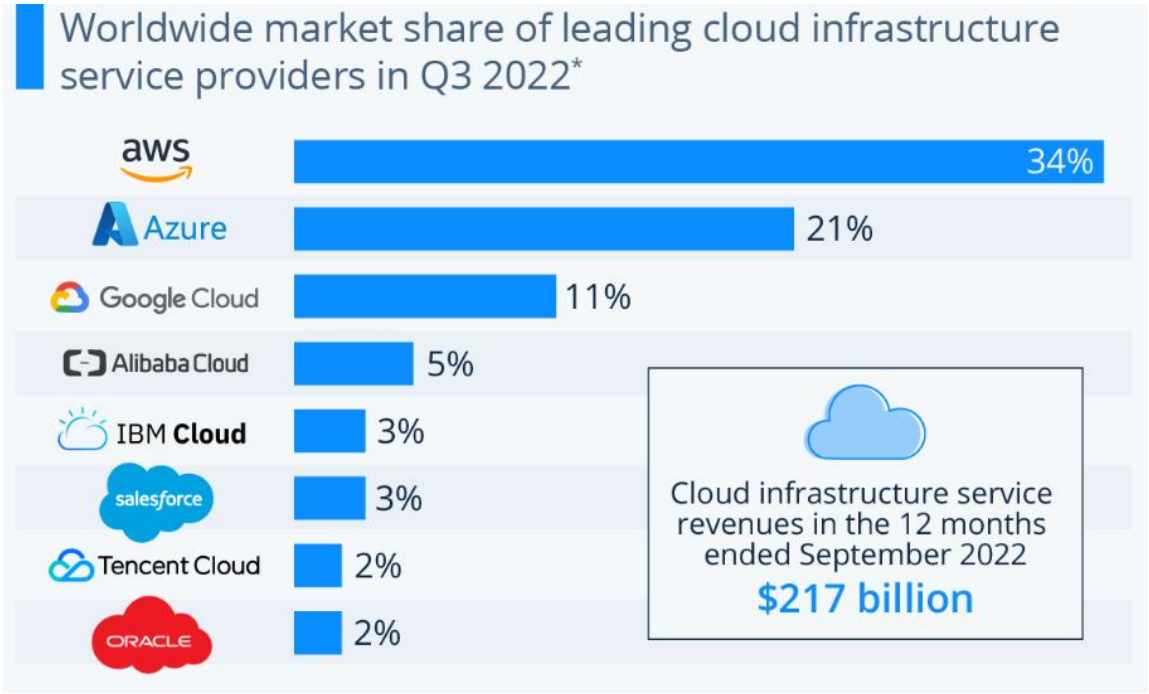

Рисунок 1.1 – Загальний дохід світових хмарних сервісів (Synergy Research Group)

Основні переваги хмарних обчислень у процесах веб-розробки полягають у наступному:

– масштабованість. Можливість масштабування, без сумніву, є найбільш важливим фактором хмарних обчислень. Незалежно від того, чи це розширення географічного охоплення або обробка зростаючої кількості користувачів, веб-додатки можна легко масштабувати, збільшуючи

кількість серверів, модернізуючи апаратну потужність або поєднуючи обидва ці способи. Це означає, що немає необхідності турбуватися про другорядні деталі, як при традиційному хостингу серверів (наприклад, конфігурація мережі, балансування навантаження на сервер і т.д.), що дозволяє забезпечити стратегічне зростання з меншою кількістю перешкод у бізнесі;

- велика кількість хмарних сервісів та ресурсів. Внутрішні сервіси та ресурси надають розробникам гнучкість для ефективної розробки, налаштування, тестування та розгортання будь-яких веб-додатків. Від автоматизованого тестування та надання серверів до веб-моніторингу та DevOps, розгортання додатків у виробничому середовищі стає простішим з майже нульовим часом простою. Крім того, показники продуктивності можна вимірювати в режимі реального часу, щоб інформувати розробників про будь-які серйозні відхилення або помилки, що виникають. Таким чином, очікувана втрата прибутку через проблеми з обслуговуванням веб-сайту різко знижується;
- економічна ефективність. Хмарні обчислення не потребують попередніх інвестицій, оскільки фізичні активи, такі як сервери та центри обробки даних, не потрібні. Оскільки ці фіксовані витрати вилучені, операційні витрати на обслуговування та встановлення такого обладнання відповідно зменшуються. Загалом, більшість хмарних сервісів надаються за принципом "pay-as-you-go". Таким чином, вартість залежить від обсягу використання, що дає можливість бізнесу починати з малого та масштабувати його за потреби. Як приклад, хмарний сервіс Google Cloud Run масштабується автоматично залежно від кількості запитів. Таким чином, він є абсолютно безкоштовним, якщо немає запитів, оскільки Cloud Run масштабується до нуля.

Хмарні обчислення можна розділити на три загальні категорії надання послуг або форми хмарних обчислень: інфраструктура як послуга (IaaS), платформа як послуга (PaaS) та програмне забезпечення як послуга (SaaS). Ці моделі пропонують

все більшу абстракцію, тому їх часто зображують як шари в стеку: інфраструктура, платформа і програмне забезпечення як послуга, але вони не обов'язково повинні бути пов'язані між собою. Наприклад, можна надавати SaaS, реалізовану на фізичних машинах, без використання базових рівнів PaaS або IaaS, і навпаки, можна запустити програму на IaaS і отримати доступ до неї безпосередньо, не обгортаючи її як SaaS.

У моделі IaaS хмарний провайдер управляє IT-інфраструктурою, такою як сховища, сервери та мережеві ресурси, і надає їх організаціям-підписникам через віртуальні машини, доступні через Інтернет. IaaS може мати багато переваг для організацій, наприклад, потенційно зробити робочі навантаження швидшими, простішими, гнучкішими та економічно ефективнішими.

У моделі послуг IaaS хмарний провайдер розміщує компоненти інфраструктури, які традиційно присутні в локальному центрі обробки даних. Сюди входять сервери, обладнання для зберігання даних і мережеве обладнання, а також рівень віртуалізації або гіпервізора.

Провайдери IaaS також надають ряд послуг, що супроводжують ці компоненти інфраструктури. Вони можуть включати наступне:

- детальне виставлення рахунків;
- моніторинг;
- доступ до журналів;
- безпека;
- балансування навантаження;
- кластеризація;
- відмовостійкість систем зберігання даних, таких як резервне копіювання, реплікація та відновлення.

IaaS дозволяє користувачам впроваджувати більш високі рівні автоматизації та оркестрування для важливих інфраструктурних завдань. Наприклад, користувач може впроваджувати політики для управління балансуванням навантаження для підтримки доступності та продуктивності додатків.

Клієнти IaaS отримують доступ до ресурсів і послуг через глобальну мережу (WAN), наприклад, Інтернет, і можуть використовувати послуги хмарного провайдера для встановлення решти елементів стека додатків. Наприклад, користувач може увійти на платформу IaaS для створення віртуальних машин (ВМ); встановити операційні системи в кожну ВМ; розгорнути проміжне програмне забезпечення, таке як бази даних; створити сховища для робочих навантажень і резервних копій; і встановити робоче навантаження підприємства в цю ВМ. Після цього клієнти можуть використовувати послуги провайдера для відстеження витрат, моніторингу продуктивності, балансування мережевого трафіку, усунення проблем з додатками та управління аварійним відновленням.

IaaS може використовуватися для широкого спектру задач. Обчислювальні ресурси, які вона надає через хмарну модель, можуть бути використані для різних варіантів використання. Найбільш поширеними варіантами використання IaaS є наступні:

- середовища тестування та розробки. IaaS пропонує організаціям гнучкість, коли мова йде про різні середовища тестування та розробки. Їх можна легко масштабувати відповідно до потреб;
- розміщення веб-сайтів, орієнтованих на клієнта. Це може зробити розміщення веб-сайту більш доступним у порівнянні з традиційними засобами розміщення веб-сайтів;
- зберігання, резервне копіювання та відновлення даних. IaaS може бути найпростішим і найефективнішим способом для організацій управляти даними, коли попит на них непередбачуваний або може постійно зростати. Крім того, організації можуть обійти необхідність великих зусиль, спрямованих на управління, юридичні та комплаєнс-вимоги до зберігання даних;
- веб-додатки. Інфраструктура, необхідна для розміщення веб-додатків, забезпечується IaaS. Тому, якщо організація розміщує веб-додаток, IaaS може надати необхідні ресурси зберігання, сервери та мережу. Розгортання

може бути здійснено швидко, а хмарна інфраструктура може бути легко збільшена або зменшена відповідно до попиту програми;

- високопродуктивні обчислення (HPC). певні робочі навантаження можуть вимагати обчислень на рівні HPC, наприклад, наукові розрахунки, фінансове моделювання та робота з проектування продуктів;
- сховища даних та аналітика великих даних. IaaS може забезпечити необхідну обчислювальну та обчислювальну потужність для обробки великих масивів даних.

Програмне забезпечення як послуга (SaaS) – це модель розповсюдження програмного забезпечення, в якій хмарний провайдер розміщує додатки і робить їх доступними для кінцевих користувачів через Інтернет. У цій моделі незалежний постачальник програмного забезпечення (ISV) може укласти контракт зі стороннім хмарним провайдером на розміщення програми. Або, у випадку з великими компаніями, такими як Microsoft, хмарний провайдер може також бути постачальником програмного забезпечення.

SaaS працює через хмарну модель доставки. Постачальник програмного забезпечення або розміщує додаток і пов'язані з ним дані, використовуючи власні сервери, бази даних, мережеві та обчислювальні ресурси, або це може бути ISV, який укладає контракт з хмарним провайдером на розміщення додатку в центрі обробки даних провайдера. Додаток буде доступним для будь-якого пристрою з підключенням до мережі. Доступ до додатків SaaS зазвичай здійснюється через веб-браузери.

Як наслідок, компанії, що використовують SaaS-додатки, не займаються налаштуванням та обслуговуванням програмного забезпечення. Користувачі просто сплачують абонентську плату, щоб отримати доступ до програмного забезпечення, яке є готовим рішенням.

SaaS тісно пов'язана з моделями надання послуг провайдера додатків (ASP) та обчислювального програмного забезпечення на вимогу, де провайдер розміщує програмне забезпечення замовника та надає його затвердженим кінцевим користувачам через Інтернет.

У моделі SaaS (програмне забезпечення на вимогу) постачальник надає клієнтам мережевий доступ до єдиної копії програми, яку він створив спеціально для розповсюдження SaaS. Вихідний код програми однаковий для всіх клієнтів, і коли з'являються нові функції або можливості, вони розгортаються для всіх клієнтів. Залежно від угоди про рівень обслуговування (SLA), дані клієнта для кожної моделі можуть зберігатися локально, в хмарі або локально і в хмарі.

Серед інших переваг моделі SaaS можна виділити:

- гнучкі платежі. Замість того, щоб купувати програмне забезпечення для встановлення або додаткове обладнання для його підтримки, клієнти підписуються на пропозицію SaaS. Перенесення витрат на поточні операційні витрати дозволяє багатьом компаніям здійснювати краще і більш передбачуване бюджетування. Користувачі також можуть у будь-який час припинити надання послуг SaaS, щоб припинити ці постійні витрати;
- масштабоване використання. Хмарні сервіси, такі як SaaS, пропонують високу вертикальну масштабованість, що дає клієнтам можливість отримати доступ до більшої або меншої кількості послуг або функцій на вимогу;
- автоматичні оновлення. Замість того, щоб купувати нове програмне забезпечення, клієнти можуть покластися на постачальника SaaS для автоматичного виконання оновлень та управління виправленнями. Це додатково зменшує навантаження на власний IT-персонал;
- доступність і стійкість. Оскільки постачальники SaaS надають додатки через Інтернет, користувачі можуть отримати доступ до них з будь-якого пристрою з доступом до Інтернету та з будь-якого місця;
- налаштування. Додатки SaaS часто налаштовуються і можуть бути інтегровані з іншими бізнес-додатками, особливо з додатками від одного постачальника програмного забезпечення.

Платформа як послуга (PaaS) – це модель хмарних обчислень, в якій сторонній постачальник надає користувачам апаратні та програмні засоби через Інтернет.

Зазвичай ці інструменти потрібні для розробки додатків. Провайдер PaaS розміщує апаратне та програмне забезпечення на власній інфраструктурі. Як наслідок, PaaS звільняє розробників від необхідності встановлювати власне обладнання та програмне забезпечення для розробки або запуску нового додатку.

PaaS не замінює всю IT-інфраструктуру компанії для розробки програмного забезпечення. Вона надається через розміщену інфраструктуру постачальника хмарних послуг. Користувачі найчастіше отримують доступ до пропозицій через веб-браузер. PaaS може надаватися через публічні, приватні та гібридні хмари для надання таких послуг, як хостинг додатків та розробка Java.

Інші послуги PaaS включають наступне:

- співпраця з командою розробників;
- проектування та розробка додатків;
- тестування та розгортання додатків;
- інтеграція веб-сервісів;
- інформаційна безпека;
- інтеграція баз даних.

Зазвичай користувачі платять за PaaS за використання. Однак деякі провайдери стягують фіксовану щомісячну плату за доступ до платформи та її додатків. Можливість здійснювати оплату на періодичній основі (підписка) або за кожне використання дозволяє підприємствам усунути капітальні витрати, які вони традиційно мають на локальне апаратне та програмне забезпечення. Фактично, PaaS перекладає відповідальність за надання, управління та оновлення ключових інструментів з внутрішньої IT-команди на зовнішнього постачальника PaaS.

Архітектура PaaS тримає свою базову інфраструктуру прихованою від розробників та інших користувачів. У результаті модель схожа на безсерверні обчислення та архітектури "функція як послуга" – це означає, що постачальник хмарних послуг керує та запускає сервер, а також контролює розподіл ресурсів.

Отже, як підсумок:

- за допомогою IaaS провайдер надає базову обчислювальну, сховищну та мережеву інфраструктуру разом з гіпервізором – рівень віртуалізації.

Потім користувачі повинні створити віртуальні екземпляри, такі як віртуальні машини та контейнери, встановити операційні системи, підтримувати програми та дані, а також обробляти всі налаштування та управління, пов'язані з цими завданнями;

- за допомогою PaaS провайдер пропонує більший стек додатків, ніж IaaS, додаючи в хмарне середовище операційні системи, проміжне програмне забезпечення, наприклад, бази даних – та інші середовища виконання;
- за допомогою SaaS постачальник пропонує цілий стек додатків. Користувачі просто входять в систему і використовують додаток, який повністю працює на інфраструктурі провайдера. Як правило, додатки SaaS повністю доступні через інтернет-браузер. Провайдери SaaS керують робочим навантаженням програми та всіма основними IT-ресурсами; користувачі лише контролюють дані, створені додатком SaaS.

Сервіси хмарних обчислень пропонують різні моделі розгортання, такі як: приватна хмара, публічна хмара, гібридна хмара.

У моделі публічної хмари сторонній постачальник хмарних послуг (CSP) надає хмарні послуги через Інтернет. Публічні хмарні послуги продаються на вимогу, як правило, похвилинно або погодинно, хоча для багатьох послуг доступні довгострокові зобов'язання. Клієнти платять лише за цикли центрального процесора, сховище або пропускну здатність, які вони споживають. Провідні публічні CSP включають AWS, Microsoft Azure, IBM і Google Cloud Platform (GCP), а також IBM, Oracle і Tencent.

Гібридна хмара – це поєднання публічних хмарних сервісів і локальної приватної хмари, з оркестровкою і автоматизацією між ними. Компанії можуть запускати критично важливі робочі навантаження або чутливі додатки в приватній хмарі, а публічну хмару використовувати для обробки сплесків навантаження або піків попиту. Мета гібридної хмари – створити уніфіковане, автоматизоване, масштабоване середовище, яке використовує переваги всього, що може надати

інфраструктура публічної хмари, зберігаючи при цьому контроль над критично важливими даними.

Крім того, організації все частіше застосовують мультихмарну модель або використання декількох провайдерів IaaS. Це дозволяє додаткам мігрувати між різними хмарними провайдерами або навіть працювати одночасно у двох або більше хмарних провайдерів.

Організації впроваджують мультихмарні технології з різних причин. Наприклад, вони можуть зробити це, щоб мінімізувати ризик відключення хмарного сервісу або скористатися перевагами більш конкурентоспроможних цін від конкретного постачальника. Впровадження мультихмарних технологій та розробка додатків може бути складним завданням через відмінності між послугами та API хмарних провайдерів.

Однак розгортання мультихмарних технологій має стати простішим, оскільки послуги та API провайдерів зближуються та стають більш стандартизованими завдяки галузевим ініціативам, таким як Відкритий інтерфейс хмарних обчислень.

Спільна хмара, яка використовується кількома організаціями, підтримує певну спільноту, яка поділяє однакові проблеми – наприклад, однакову місію, політику, вимоги безпеки та міркування щодо відповідності. Хмара спільноти управляється або цими організаціями, або стороннім постачальником і може знаходитися в приміщенні або за його межами.

Одним із загальновживаних термінів в екосистемі хмарних обчислень є термін "інстанс".

Інстанс у хмарних обчисленнях – це не що інше, як віртуальний сервер, на якому запущено додаток. Інстанс також називають віртуальною машиною або сервером, але в екосистемі хмарних обчислень термін "інстанс" завжди має перевагу над іншими.

Ці інстанс надаються відповідно до потреб користувачів, і вони оновлюються або знижуються за допомогою хмарного програмного забезпечення. Крім того, ці інстанс можуть бути створені на вимогу і можуть бути зупинені, якщо вони не потрібні.

Користувачі можуть використовувати один або декілька інстанс для виконання своїх завдань. Вони також можуть групувати кілька інстанс для своїх завдань. Ці інстанс також можуть бути створені в різних географічних регіонах по всьому світу.

Інстанс мають заздалегідь визначену базову конфігурацію безпеки та мережеву конфігурацію від хмарного провайдера, і користувачам дозволяється додавати власні політики конфігурації відповідно до їхніх потреб.

Ці інстанс можуть бути створені вручну за допомогою хмарного програмного забезпечення або можуть бути автоматизовані за допомогою API або SDK хмарного провайдера.

Для прикладу, розглянемо інстанс у AWS, відомий як EC2. Веб-сервіс Amazon (AWS) надає багато різних інстанс для різних робочих навантажень (рис. 1.2). Ці інстанс надаються через послугу, яка називається Elastic Compute (EC2).

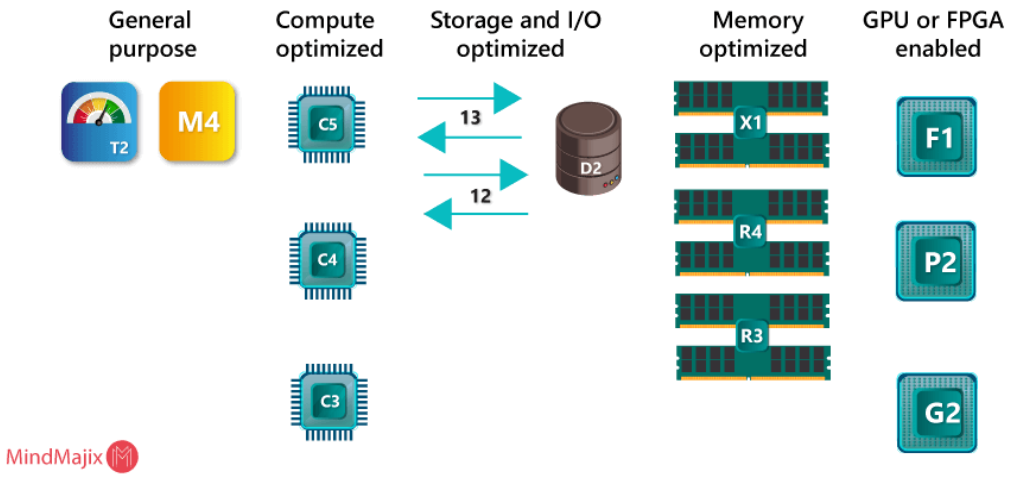

Рисунок 1.2 – Різні категорії інстанс у AWS

1. Інстанс загального призначення. Інстанс загального призначення ідеально підходять для стандартного використання, наприклад, для веб-серверів, ці інстанс мають рівну частку обчислювальної потужності, пам'яті та мережевих ресурсів. Наприклад, інстанс загального призначення можна

використовувати для розгортання простих і середніх веб-серверів. Він добре підходить для:

- сервери додатків;
- внутрішні додатки організації;
- сторонні / університетські проекти ;
- простих кеш-серверів.

2. Оптимізований для обчислень інстанс. Оптимізований для обчислень інстанс ідеально підходить для робочих навантажень, які потребують високої обчислювальної потужності. Ці ресурси оптимізовані для забезпечення високої обчислювальної потужності. Наприклад, оптимізований для обчислень інстанс можна використовувати для додатків, які обробляють величезну кількість наборів даних для прогнозування результатів. Він добре підходить для:
   - науково-дослідницьких проектів;
   - складних проектів в області науки про дані;
   - ігрових серверів;
   - пакетних процесорів обробки даних.
3. Інстанс прискорених обчислень. Інстанс прискорених обчислень надає апаратні прискорювачі і підходить для додатків, які вимагають інтенсивного графічного рендерингу або аналізу величезної кількості візуальних даних, таких як зображення або відео. Наприклад, інстанс прискорених обчислень може використовуватися для складних моделей машинного навчання. Він добре підходить для
   - графічного рендерингу;
   - обробки зображень і відео;
   - складних аналітичних навантажень.
4. Інстанс з оптимізацією пам'яті. Оптимізовані по пам'яті інстанс ідеально підходять для розгортання додатків, що вимагають багато пам'яті, без шкоди для продуктивності. Ці екземпляри забезпечують високу продуктивність для додатків, які вимагають обробки великих масивів даних

в пам'яті. Наприклад, інстанс з оптимізованою пам'яттю застосовуються в аналітиці великих даних. Добре підходять для:

- високопродуктивних баз даних;
- аналітики великих даних;
- розподілені кеші в пам'яті.

5. Оптимізований інстанс сховища. Оптимізований інстанс сховища налаштований для збільшення стійкості сховища для читання і запису даних на дуже високій швидкості. Він ідеально використовується для баз даних, які вимагають високої швидкості читання і запису. Він добре підходить для:
   - індексаторів баз даних;
   - функції пошукових систем.

Хмарні інженери приймають рішення про те, який екземпляр підходить відповідно до робочого навантаження. Дуже важливо переконатися, що при виборі екземпляра враховується кожен аспект вимог, цим часто займається сертифікований архітектор хмари.

Інстанс з моменту створення проходить різні етапи свого життєвого циклу (рис. 1.3). Основні його етапи: надання, очікування, виконання, зупинення, перезапуск, припинення.

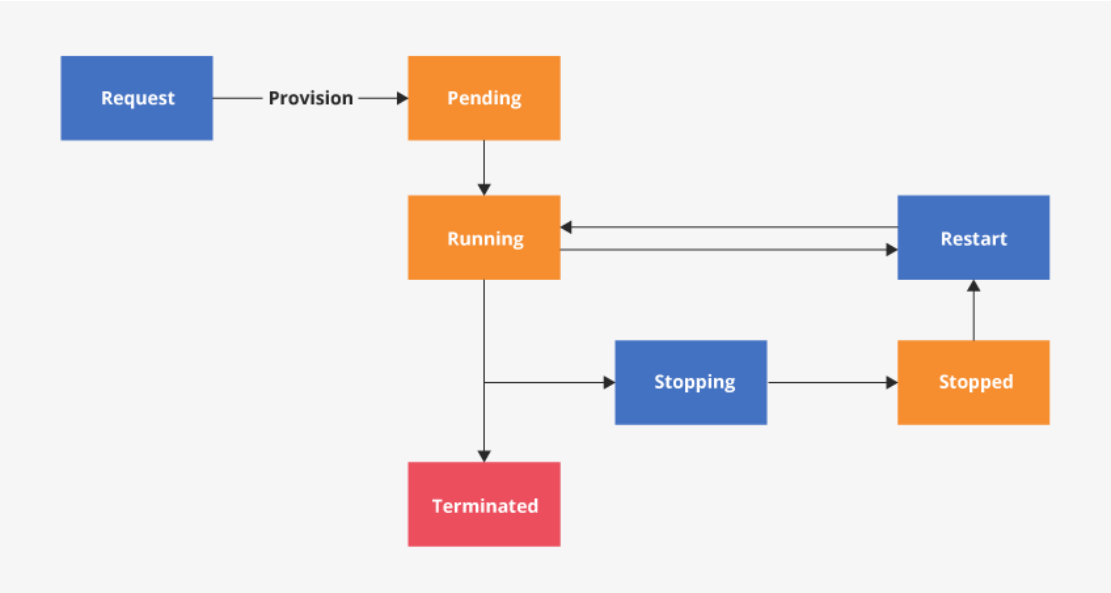

Рисунок 1.3 – Життєвий цикл інстанс

1. Надання. Це етап, на якому користувачі надсилають запит на інстанс відповідно до своїх потреб. Після цього хмара забезпечить надання інстанс відповідно до потреб користувача.
2. Очікування. Після надання інстанс переходить в стан очікування, щоб можна було застосувати конфігурацію безпеки і мережі, перш ніж він буде готовий. Інстанс також переходить в стан очікування, коли користувач змінює вимоги до істанс, щоб ці зміни могли бути застосовані.
3. Запуск. Інстанс буде функціональним і готовим до використання, коли він знаходиться в стані виконання. Інстанс переходить у стан виконання тільки тоді, коли застосовані всі необхідні конфігурації. Коли інстанс знаходиться в запущеному стані, користувачі можуть почати взаємодіяти з ним.
4. Зупинений. Якщо користувачеві необхідно зупинити роботу інстанс на короткий проміжок часу, він може вимкнути інстанс. Це дозволить змінити налаштування або, можливо, налагодити деякі проблеми, пов'язані з додатком. Користувачам не будуть виставлятися рахунки за зупинений інстанс.

5. Перезапуск. Інстанс переходить в стадію перезапуску, коли користувачі вручну перезапускають інстанс або коли користувачі вирішують запустити зупинені інстанс.
6. Припинення. Інстанс переходить у стадію припинення, коли користувачі вирішують ліквідувати інстанс після його використання. Завершення призводить до видалення відповідних даних в томі сховища інстанс.

Хмарний інстанс дозволяє людям масштабуватися за межі традиційних фізичних кордонів. За допомогою хмари людям не потрібно турбуватися про базове обладнання при розгортанні робочих навантажень на хмарному інстанс. Переваги хмарних інстанс:

1. масштабованість. Хмарні інстанс дозволяють користувачам масштабувати обчислювальні ресурси відповідно до вимог робочого навантаження. Наприклад, якщо додаток отримує більше користувачів, він відчуває величезний трафік, який уповільнює час відгуку, тому, збільшуючи хмарні ресурси, такі як процесор, пам'ять, сховище та мережеві ресурси для конкретного інстанс, можна легко впоратися зі сплеском трафіку;
2. обслуговування інфраструктури. Хмарний інстанс дозволяє користувачам більше зосередитися на своїх додатках та вимогах, а не на інфраструктурі. За допомогою хмарного інстанс користувачі можуть отримати віртуальну машину на вимогу і почати використовувати її, замість того, щоб встановлювати фізичний сервер і займатися постачанням обладнання;
3. безпека. Хмарний інстанс забезпечує безпеку в багатьох аспектах, таких як шифрування для захисту будь-яких даних, які він обробляє, захист доступу, авторизація, щоб дозволити тільки авторизованим користувачам отримати доступ до інстанс для його оновлення та модифікації;
4. відмовостійкість. Користувачі можуть використовуючи багато дублікатів інстанс для резервного копіювання. Вони особливо корисні для управління робочими навантаженнями, що вимагають багато пам'яті.

Таблиця 1.1 – Класи, родини та типи інстанс у AWS

Типи інстанс Amazon EC2				
Загальне призначення	Оптимізовані обчислення	Оптимізована пам’ять	Оптимізовано сховище	Графіні обчислення
Mac, T, M, A	C, Hpc	R, X, High Memory, Z	I, D, H	P, DL, Trn, Inf, G, F, VT
Mac T2 T3a T3 T4g M4 M5zn M5n M5a M5 M6a M6i M6g A1	C4 C5n C5a C5 Hpc6a C6a C6i C6gn C6g C7g	R4 R5n R5b R5a R5 R6i R6g R6a X1 X1e X2iezn X2iedn X2idn X2gd High Memory	I3en 13 I4i Is4gen Im4gn D2 D3 D3en H1	P2 P3 P4 Inf1 Trn1 DL1 G3 G4ad G4dn G5g G5 F1 VT1

Кожна з категорій класів інстанс поділяється на різні розміри та сімейства інстанс EC2. Тип інстанс EC2 описує комбінацію процесора, оперативної пам'яті, сховища та мережевих можливостей конкретного сімейства інстанс. У таблиці 1.1 висвітлено різні типи інстанс.

Для кожного типу екземплярів EC2 пропонує до семи розмірів інстанс: Nano, Micro, Mini, Medium, Large, Xlarge, 2Xlarge.

Знову ж таки, мета гнучкості – переконатися в отриманні достатньої обчислювальної потужності для робочого навантаження, щоб заощадити гроші без шкоди для продуктивності. EC2 також автоматично поєднує кожен інстанс з відповідним процесором, оперативною пам'яттю, сховищем та мережевою ємністю.

Для прикладу розберемо тип інстансу T4g.

Це нове покоління екземплярів сімейства T, яке оснащене власними процесорами Graviton 2 (64-розрядними) на базі Arm від AWS, які забезпечують до 40% кращу продуктивність за ціною в порівнянні з екземплярами T3. Нижче наведено таблицю порівняння інстанс T4g різного розміру.

Таблиця 1.2 = Порівняння інстанс T4g різного розміру

T4 Розмір	**vЦП**	**Пам’ять у Гб**	**Мережева швидкість**	**Мінімальна продуктивність для отримання кредитів**	**Згенеровані кредити за годину роботи**
t4.nano	2	0.5	Up to 5Ghz	5%	6
t4.micro	2	1	Up to 5Ghz	10%	12
t4.small	2	2	Up to 5Ghz	20%	24
t4.medium	2	4	Up to 5Ghz	20%	24
t4.large	2	8	Up to 5Ghz	30%	36
t4.xlarge	4	16	Up to 5Ghz	40%	96
t4.2xlarge	8	32	Up to 5Ghz	40%	192

Тип T4g використовується у бізнес-критичних додатках, мікросервісах, малих/середніх базах даних та середовищ розробки, а також віртуальних робочих столах.

1.2 Проблема оцінки вартості розгортання додатку у хмарному середовищі

Існує три основні компоненти, які визначають вартість послуг хмарних обчислень:

1. обчислювальні ресурси – більшість хмарних провайдерів пропонують ряд типів обчислювальних інстанс, кожен з яких пропонує певну кількість ресурсів процесора, пам'яті, а в деяких випадках і спеціалізоване обладнання, таке як швидкісні мережеві або графічні прискорювачі. Клієнт платить відповідно до кількості та типів використовуваних інстанс і тривалості їх використання;
2. мережеві – більшість хмарних сервісів виставляють рахунки клієнтам відповідно до обсягу даних, переданих до хмарного сервісу (вхід), з хмарного сервісу (вихід), або і того, і іншого. За віртуалізовані мережеві послуги, такі як статичні IP-адреси, балансувальники навантаження та шлюзи, може стягуватися додаткова плата;
3. сховище – провайдери хмарних сховищ пропонують сховище як послугу. За послуги еластичного зберігання клієнти платять за фактично використаний гігабайтний місяць зберігання. За послуги керованого зберігання, такі як керовані диски, підключені до обчислювальних інстанс, клієнти платять за весь обсяг сховища, незалежно від обсягу використаного сховища на цьому інстанс.

Порівняємо витрати на хмарні технології з традиційною інфраструктурою. Ось три основні категорії витрат, які зазвичай пов'язані зі створенням і підтримкою локальної інфраструктури:

- капітальні витрати – серверне програмне забезпечення, ліцензування та обладнання, а також мережева інфраструктура, середовища зберігання та системи резервного копіювання;
- операційні витрати – включаючи підтримку серверної та мережевої інфраструктури, а також гарантійні зобов'язання на зберігання даних,

послуги центрів обробки даних, витрати на оплату праці існуючого системного адміністрування, а також навчання утримання IT-персоналу;

- непрямі бізнес-витрати, включаючи незаплановані та заплановані простої. Зазвичай, на кожен долар, витрачений на модернізацію інфраструктури, організації можуть розраховувати витратити приблизно 2 долари на управління, обслуговування та забезпечення безпеки розширеної інфраструктури. При розрахунку загальних витрат на міграцію та впровадження хмарних технологій організаціям необхідно спочатку провести комплексний аудит існуючих IT, включаючи як прямі, так і непрямі витрати;
- прямі витрати – включають апаратне та програмне забезпечення, управління, технічне обслуговування та персонал, а також будь-які фізичні об'єкти. Прямі витрати, як правило, прості та легко піддаються оцінці.

Непрямі витрати включають втрату продуктивності, яка може бути наслідком декількох факторів, таких як простій сервера, втрата довіри клієнтів та шкода репутації. Непрямі витрати, як правило, складніше передбачити та оцінити.

Нижче наведено кілька поширених моделей ціноутворення, що використовуються в хмарі, які можна комбінувати в залежності від потреб:

- оплата за фактом використання. У цій моделі рахунки за хмарні сервіси виставляються за фактичне використання. Хмарні сервіси можуть виставляти рахунки за використання обчислювальних потужностей, сховища, мережі або інших ресурсів. Перевага полягає в тому, що оплата передбачається тільки за фактичне використання і можна зменшити обсяг ресурсів, коли це необхідно. Недоліком є те, що при додаванні ресурсів до хмарного розгортання, поточні витрати можуть швидко зрости;
- передоплата/фіксована підписка. У моделі, що базується на підписці, клієнти хмари платять за послуги заздалегідь. Ціни на підписку передбачають надання заздалегідь визначеного пакету послуг протягом певного часу. Чим довший період, тим нижча ціна. Ціноутворення на основі підписки є поширеним для хмарних сервісів, які поєднують декілька

апаратних та програмних елементів, таких як платформа як послуга (PaaS) та програмне забезпечення як послуга (SaaS). Більшість хмарних провайдерів також пропонують ціни на основі підписки для клієнтів з високими витратами, що дозволяє їм укласти корпоративний дисконтний план, де вони зобов'язуються до певного рівня витрат на хмару та отримують знижку на деякі або всі свої хмарні послуги;

- зарезервовані інстанс. Зарезервовані інстанс дозволяють компаніям взяти на себе зобов'язання щодо хмарних ресурсів на тривалий період часу, як правило, на 1 або 3 роки. Чим довший термін, і чим більше компанія готова внести передоплати на початку періоду, тим більша знижка. Трирічний термін зазвичай є найбільш економічно вигідним. Хмарні провайдери зазвичай пропонують знижки в розмірі 50-75% в порівнянні з оплатою по факту на зарезервовані інстанс з тими ж можливостями. Зарезервовані інстанс підходять для стабільних навантажень та довготривалих систем. Однак організаціям не слід використовувати зарезервовані екземпляри для пікових навантажень. Замість цього зарезервовані потужності повинні використовуватися для основних компонентів системи, а додаткові потужності, необхідні під час пікових навантажень, повинні оброблятися за допомогою інстанс з оплатою за фактом використання або спотових інстанс;
- план заощадження AWS. Подібно до зарезервованих екземплярів, плани заощадження – це гнучка модель ціноутворення, яка дозволяє організаціям користуватися нижчими цінами, ніж за тарифом «на вимогу», в обмін на річне або трирічне зобов'язання щодо конкретного використання. Зобов'язання виражається у витратах за годину на послуги Amazon. AWS пропонує три типи ощадних планів:
  - Savings Plans – застосовуються до всіх випадків використання обчислювальних сервісів Amazon, включаючи EC2, AWS Lambda та Fargate;

- EC2 Savings Plans – застосовується лише до використання інстанс Amazon EC2;
- SageMaker Savings Plans – стосується лише використання SageMaker.

План заощадження пропонує три способи оплати:

- без авансового платежу – не вимагає авансового платежу, рахунки виставляються клієнтам відповідно до фактичного використання щомісяця. Це надає мінімальну знижку за ощадним планом;
- частковий авансовий платіж – при цьому варіанті більше половини контракту оплачується авансом, а решта виставляється щомісяця, що надає додаткову знижку;
- повний авансовий платіж – повна сума сплачується авансом, що надає найбільшу знижку;

– спотові інстанс. Як правило, є найдешевшим варіантом обчислень, пропонуючи знижки до 90% порівняно з тарифами, що оплачуються в міру використання. Спотові інстанс використовуються хмарними провайдерами для розпродажу вільних потужностей. Знижка надається з урахуванням того, що спотові інстанс можуть бути перервані в найкоротші терміни. Зазвичай спотові інстанс можна використовувати лише для робочих навантажень, які є відмовостійкими або для процесів, які можна зупиняти та перезапускати.

Управління витратами на хмарні технології (або оптимізація витрат на хмарні технології) дозволяє бізнесу централізовано управляти витратами, пов'язаними з хмарними технологіями, максимізувати віддачу від інвестицій у хмарні ресурси та підвищити ефективність.

Хмарні ресурси є дуже динамічними і можуть бути розподілені між різними місцями, сервісами та хмарними провайдерами. Легка масштабованість і розгортання є переконливими перевагами хмарних технологій, але вони також дозволяють IT, розробникам або інженерним командам легко управляти ресурсами, не звертаючи уваги на короткострокові або довгострокові витрати. Хмара ускладнює досягнення прозорості та прийняття рішень і політик щодо витрат на

хмарні технології. Стратегія управління витратами на хмарні технології може допомогти організаціям планувати майбутнє споживання та витрати на хмарні сервіси.

Сьогодні більшість компаній використовують декілька хмар, що робить критично важливим прийняття стратегій управління витратами на декілька хмар.

Розглянемо деякі зі стратегій, які компанії можуть використовувати для управління витратами на хмарні технології:

- бюджетний контроль – у якості першого кроку організації повинні встановити бюджети на хмарні сервіси і переконатися, що команди знають про них і не можуть перевищувати бюджет для свого конкретного проекту;
- правильне визначення розміру – забезпечення інстанс обчислень, обсягів зберігання та інших послуг на тому рівні, який дійсно необхідний бізнесу. Дуже часто хмарні ресурси надаються, але не використовуються в повному обсязі;
- автомасштабування – динамічно збільшує та зменшує ресурси відповідно до попиту додатків, гарантуючи, що оплата за додаткові хмарні ресурси відбудеться лише під час пікового використання;
- планування – багато хмарних сервісів не вимагають цілодобової роботи 24/7, і їх можна запланувати на вимкнення, коли вони не потрібні. Наприклад, служби, що використовуються командою, яка базується в США, можуть бути вимкнені в неробочий час;
- виявлення невикористаних ресурсів – обчислювальні інстанс, томи сховищ, балансувальники навантаження, о́брази і багато інших ресурсів можуть бути легко створені і забуті. Організації повинні мати можливість сканувати своє хмарне розгортання на наявність невикористаних ресурсів і видаляти їх для економії витрат;
- розумне застосування моделей ціноутворення зі знижками, таких як зарезервовані інстанс та спотові інстанс, може значно знизити витрати на хмару, але вони повинні використовуватися належним чином.

На відміну від традиційних IT-систем, які були добре відомі та статичні, управління витратами за хмарні сервіси не може легко здійснюватися за допомогою електронних таблиць та ручних списків. Існують автоматизовані інструменти, які можуть отримувати метрики з API, звітувати про споживання хмарних послуг та витрати на них, а також вносити зміни до послуг у разі необхідності. Існує два основних типи інструментів – інструменти першої сторони, що надаються безпосередньо хмарним провайдером, та інструменти третьої сторони від зовнішніх постачальників.

Всі загальнодоступні хмарні платформи надають інструменти управління витратами. Ці інструменти тісно інтегровані з хмарною платформою і доступні "з коробки" без особливих зусиль по розгортанню. Деякі з цих інструментів є безкоштовними у використанні, тоді як інші оплачуються за моделлю оплати за використання.

У багатьох випадках ці базові інструменти є найшвидшим способом розпочати управління витратами на хмарні технології. Однак вони мають ряд обмежень, які вирішуються за допомогою сторонніх інструментів:

- обмежена функціональність – більшість сторонніх інструментів обмежені в своїй здатності виявляти марні витрати і максимізувати економію, використовуючи кілька моделей витрат;
- обмеженість одним постачальником – більшість сторонніх інструментів працюють лише з одним хмарним провайдером і не підходять для організацій, які потребують управління витратами в декількох хмарах;
- конфлікт інтересів – хмарний провайдер, в кінцевому рахунку, зацікавлений в максимізації прибутку. З одного боку, хмарні провайдери дійсно хочуть допомогти клієнтам запускати додатки економічно ефективно, збільшити використання та утримання. З іншого боку, хмарні провайдери хочуть максимізувати споживання хмарних послуг і не завжди можуть запропонувати оптимальне рішення для клієнта.

Сторонні інструменти управління витратами можуть усунути функціональні обмеження інструментів першої сторони і, як правило, забезпечують мультихмарне

управління витратами. Більшість з цих інструментів створені для зниження витрат на хмарні сервіси та робочі навантаження, забезпечуючи чітку рентабельність інвестицій (ROI).

Однак організаціям слід ретельно оцінити функції інструментів управління хмарними технологіями в різних хмарах, зрозуміти, як їх можливості співвідносяться з конкретними моделями витрат кожної хмари, а також оцінити їх здатність виявляти, рекомендувати і автоматизувати оптимізацію витрат.

Ще однією проблемою в оцінці вартості розгортання програми в хмарі є динамічний характер хмарних середовищ. Хмарні ресурси, як правило, виділяються на основі оплати за фактом, що означає, що вартість їх використання може коливатися в залежності від таких факторів, як моделі використання та попит. Це може ускладнити прогнозування довгострокової вартості розгортання програми в хмарі, оскільки несподівані сплески використання або зміни попиту можуть суттєво вплинути на загальну вартість.

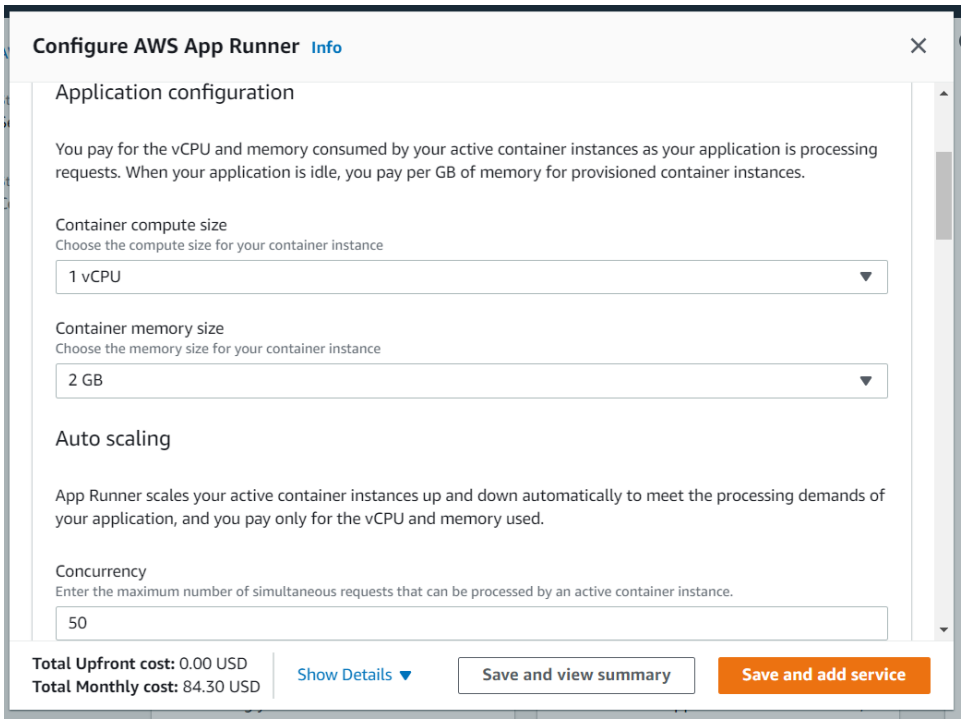

Рисунок 1.4 – Приклад розрахунків калькулятора цін AWS за моделлю «оплата за фактом використання»

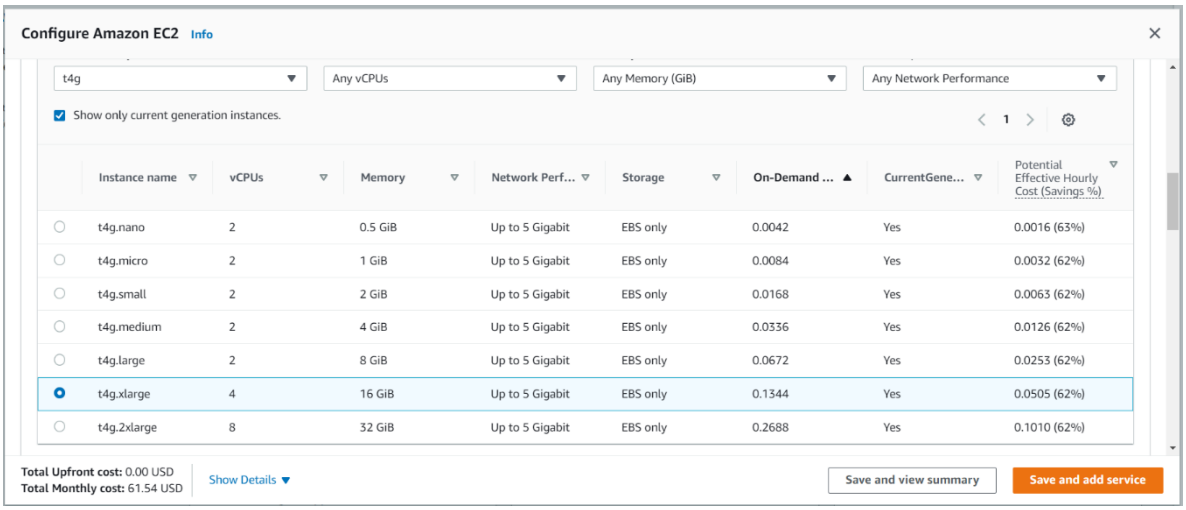

Рисунок 1.5 – Приклад розрахунків калькулятора цін AWS для спотового інстансу Ec2

Для того, щоб точно оцінити вартість розгортання програми в хмарі, важливо ретельно розглянути всі фактори, які можуть вплинути на кінцеву вартість. Це включає в себе визначення конкретних послуг і ресурсів, необхідних для підтримки програми, розуміння моделей ціноутворення і умов надання послуг, пропонованих різними хмарними провайдерами, а також врахування будь-яких потенційних коливань у використанні або попиті. Ретельно розглядаючи ці фактори і використовуючи такі інструменти і ресурси, як калькулятори цін на різні види послуг та моделей ціноутворення (рис. 1.4, 1.5), організації можуть більш точно оцінити вартість розгортання своїх додатків в хмарі.

Але однією з ключових, проте ще не охоплених проблем оцінки вартості розгортання програми в хмарі, є величезна різноманітність доступних хмарних провайдерів і моделей ціноутворення. Кожен провайдер пропонує унікальний набір послуг і структур ціноутворення, що ускладнює порівняння витрат і визначення найбільш економічно вигідного варіанту. Крім того, моделі ціноутворення, що використовуються хмарними провайдерами, можуть бути складними і важкими для розуміння, з широким спектром змінних і факторів, які можуть вплинути на кінцеву

вартість. А, отже, на вибір провайдеру сервісу та типу послуги буде витрачено багато ресурсів.

1.3 Постановка задачі

Для вирішення проблеми оцінки вартості розгортання додатку у хмарному середовищі необхідно реалізувати інформаційну технологію агрегатора сервісів провайдерів хмарних технологій. На основі критеріїв, заданих користувачем, агрегатор має надати найкращі пропозиції за ціною з усіх комбінацій моделей ціноутворення, типів послуг, сервісів провайдерів хмарних технологій.

У рамках кваліфікаційної роботи буде реалізовано функціонал визначення найкращих цін за типами послуг "за вимогою" та "зарезервований інстанс", різних типів інстанс з урахуванням гнучких параметрів пікових навантажень таких сервісів провайдерів хмарних послуг:

- AWS App Runner;
- GCP App Engine;
- AWS EC2;
- Azure App Service;
- Alibaba Simple Application Server.

Висновки до розділу 1

У розділі описано та проаналізовано існуючі моделі хмарних обчислень, визначено основні фактори, які впливають на вартість розгортання додатку у хмарі. Також розглянуто проблеми оцінки вартості розгортання додатку у хмарному середовищі. Так як величезна різноманітність доступних хмарних провайдерів і моделей ціноутворення ускладнює порівняння витрат і визначення найбільш економічно вигідного варіанту, то для вирішення проблеми необхідно реалізувати інформаційну технологію агрегатора сервісів провайдерів хмарних технологій, який на основі критеріїв, заданих користувачем, надає найкращі пропозиції за ціною з

усіх комбінацій моделей ціноутворення, типів послуг, сервісів провайдерів хмарних технологій.

2. РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ОЦІНКИ ВАРТОСТІ РОЗГОРТАННЯ ПРОГРАМНОГО ДОДАТКУ У ХМАРНОМУ СЕРЕДОВИЩІ

2.1 Вибір підходів та засобів для реалізації інформаційної технології

Одним з найважливіших компонентів інформаційної технології є база даних, оскільки у ній зберігаються усі дані, якими оперує інформаційна система. Є кілька аспектів, які слід враховувати при виборі правильної моделі даних і бази даних. Ці аспекти залежать від етапу життєвого циклу даних, для якого розробляється модель. Існують наступні фактори:

- швидкість і частота створення та модифікації даних – невеликі обсяги даних повинні записуватися на диск швидше, зберігаючи при цьому узгодженість;
- швидкість отримання даних – малі або великі обсяги даних, які необхідно отримати для звітності та аналізу;
- властивості ACID – атомарність, узгодженість, ізольованість і довговічність транзакцій;
- сфера діяльності – один або декілька відділів або бізнес-функцій;
- доступ до найнижчого рівня даних – різні варіанти використання даних можуть вимагати доступу до найнижчого рівня деталізації або різних рівнів агрегації.

Можуть бути й інші фактори, але згадані вище значною мірою впливають на прийняття рішення щодо вибору правильної моделі даних і на її основі вибору бази даних.

При розробці необхідно, перш за все, спроектувати такий програмний продукт, який задовольняв би чотирьом принципам: ефективність, контроль, сумісність, гнучкість.

На даний момент реляційна модель використовується найчастіше.

Реляційна база даних – це сукупність інформації, яка організовує точки даних з визначеними взаємозв'язками для легкого доступу до них. У моделі реляційної бази даних структури даних, включаючи таблиці даних, індекси та подання, залишаються відокремленими від фізичних структур зберігання, що дозволяє адміністраторам баз даних редагувати фізичне сховище даних, не впливаючи на логічну структуру даних.

На підприємствах реляційні бази даних використовуються для організації даних та визначення взаємозв'язків між ключовими точками даних. Вони дозволяють легко сортувати і знаходити інформацію, що допомагає організаціям приймати бізнес-рішення більш ефективно і мінімізувати витрати. Вони добре працюють зі структурованими даними.

Таблиці даних, що використовуються в реляційній базі даних, зберігають інформацію про пов'язані об'єкти. Кожен рядок містить запис з унікальним ідентифікатором – відомим як ключ – і кожен стовпчик містить атрибути даних. Кожен запис присвоює значення кожному атрибуту, що дозволяє легко ідентифікувати зв'язки між точками даних.

Стандартним інтерфейсом користувача та прикладного програмного забезпечення (API) реляційної бази даних є структурована мова запитів (Structured Query Language). Оператори коду SQL використовуються як для інтерактивних запитів інформації з реляційної бази даних, так і для збору даних для звітів. Для забезпечення точності та доступності реляційної бази даних необхідно дотримуватися визначених правил цілісності даних.

До ключових переваг реляційних баз даних можна віднести наступні:

– категоризація даних. Адміністратори баз даних можуть легко класифікувати та зберігати дані в реляційній базі даних, які потім можна запитувати та фільтрувати для вилучення інформації для звітів. Реляційні бази даних також легко розширювати і вони не залежать від фізичної організації. Після створення початкової бази даних можна додати нову категорію даних без необхідності модифікації існуючих додатків;

- точність. Дані зберігаються лише один раз, що виключає дедуплікацію даних в процедурах зберігання;
- простота використання. Складні запити легко виконуються користувачами за допомогою SQL – основної мови запитів, що використовується в реляційних базах даних;
- спільна робота. До однієї і тієї ж бази даних можуть звертатися кілька користувачів;
- безпека. Безпосередній доступ до даних в таблицях всередині СКБД може бути обмежений для конкретних користувачів.

До недоліків реляційних баз даних можна віднести наступні:

- структура. Реляційні бази даних вимагають багато структури і певного рівня планування, оскільки стовпці повинні бути визначені, а дані повинні правильно вписуватися в дещо жорсткі категорії. Структура хороша в деяких ситуаціях, але вона створює проблеми, пов'язані з іншими недоліками, такими як обслуговування і відсутність гнучкості та масштабованості;
- проблеми з обслуговуванням. Розробники та інший персонал, відповідальний за базу даних, повинні витрачати час на управління та оптимізацію бази даних по мірі того, як до неї додаються дані;
- негнучкість. Реляційні бази даних не є ідеальними для обробки великих обсягів неструктурованих даних. Дані, які в значній мірі є якісними, важко визначеними або динамічними, не є оптимальними для реляційних баз даних, оскільки зі зміною або розвитком даних схема повинна розвиватися разом з ними, що потребує часу;
- відсутність масштабованості. Реляційні бази даних погано масштабуються по горизонталі у фізичних структурах зберігання з декількома серверами. Важко працювати з реляційними базами даних на декількох серверах, тому що, коли набір даних стає більшим і більш розподіленим, структура порушується, а використання декількох серверів впливає на продуктивність, наприклад, час відгуку додатків, і доступність.

Реляційні бази даних працюють зі структурованими даними з визначеними взаємозв'язками, які можуть бути організовані в табличному форматі. Однак вибір правильної архітектури бази даних – це набагато більше, ніж просто вибір між реляційною та нереляційною. Тип даних і додатки, що використовуються або розробляються, є ключовими факторами, які слід враховувати.

Певні ініціативи вимагають особливих міркувань при виборі програмного забезпечення для баз даних. Наприклад, в ініціативах IoT питанням є вибір між SQL і NoSQL, а також між статичними і потоковими базами даних.

З доступних варіантів вимоги інформаційної технології найкраще задовольняє саме реляційна модель.

За даними порталу db-engines.com найпопулярнішими RDBMS є Oracle, Mysql, Microsoft SQL Server та PostgreSQL (рис. 2.1).

399 systems in ranking, December 2022

Rank			DBMS	Database Model	Score		
Dec 2022	Nov 2022	Dec 2021			Dec 2022	Nov 2022	Dec 2021
1.	1.	1.	Oracle	Relational, Multi-model	1250.31	+8.62	-31.43
2.	2.	2.	MySQL	Relational, Multi-model	1199.40	-6.14	-6.64
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	924.35	+11.84	-29.67
4.	4.	4.	PostgreSQL	Relational, Multi-model	617.97	-5.18	+9.76
5.	5.	5.	MongoDB	Document, Multi-model	469.33	-8.57	-15.34
6.	6.	6.	Redis	Key-value, Multi-model	182.57	+0.52	+9.03
7.	↑ 8.	7.	IBM Db2	Relational, Multi-model	146.61	-2.95	-20.56
8.	↓ 7.	8.	Elasticsearch	Search engine, Multi-model	144.93	-5.40	-12.80
9.	9.	↑ 10.	Microsoft Access	Relational	133.83	-1.20	+7.84
10.	10.	↓ 9.	SQLite	Relational	132.44	-2.19	+3.76

Рисунок 2.1 – Рейтинг популярності

Oracle DB. Корпорація Oracle володіє базою даних Oracle, і її код не є відкритим. СУБД Oracle призначена для великих додатків, особливо в банківській сфері. Більшість провідних банків світу використовують додатки Oracle, оскільки Oracle пропонує потужну комбінацію технологій та комплексних, попередньо інтегрованих бізнес-додатків, включаючи необхідну функціональність, створену спеціально для банків.

Основним недоліком використання Oracle є те, що вона не є безкоштовною у використанні, як її конкуренти з відкритим вихідним кодом, і може бути досить дорогою.

MySQL є найпопулярнішою базою даних SQL з відкритим вихідним кодом. Вона зазвичай використовується для розробки веб-додатків. Основними перевагами MySQL є те, що вона проста у використанні, недорога, надійна (існує з 1995 року) і має велику спільноту розробників, які можуть допомогти відповісти на питання.

Деякі з недоліків полягають у тому, що вона, як відомо, страждає від низької продуктивності при масштабуванні, розробка з відкритим вихідним кодом відстала з тих пір, як Oracle взяла під контроль MySQL, і вона не включає в себе деякі розширені функції, до яких розробники можуть звикнути.

SQL Server. Microsoft є власником SQL Server. Як і у випадку з СУБД Oracle, код є закритим. Великі корпоративні додатки в основному використовують SQL Server. Microsoft пропонує безкоштовну версію початкового рівня під назвою Express, але вона може стати дуже дорогою при масштабуванні додатку.

PostgreSQL – це база даних SQL з відкритим вихідним кодом, яка не контролюється жодною корпорацією. Зазвичай вона використовується для розробки веб-додатків.

PostgreSQL має багато з тих самих переваг, що й MySQL. Вона проста у використанні, недорога, надійна і має велику спільноту розробників. Вона також надає деякі додаткові можливості, такі як підтримка зовнішніх ключів, не вимагаючи складної конфігурації.

Основним недоліком PostgreSQL є те, що вона може бути повільнішою в роботі, ніж інші бази даних, такі як MySQL. Вона також трохи менш популярна, ніж MySQL.

Отже для інформаційної технології найкраще підходить СУБД MySQL.

2.2 Концептуальна модель оцінки вартості розгортання програмного додатку у хмарному середовищі

Моделювання даних (data modelling) – це процес створення моделі даних для даних, які будуть зберігатися в базі даних. Ця модель є концептуальним представленням об'єктів даних, асоціацій між різними об'єктами даних та правил.

Моделювання даних допомагає у візуальному представленні даних та забезпечує дотримання бізнес-правил, нормативних вимог та політики щодо даних. Моделі даних забезпечують узгодженість в угодах про імена, значень за замовчуванням, семантиці, безпеці, забезпечуючи при цьому якість даних.

Модель даних визначається як абстрактна модель, яка організовує опис даних, семантику даних та обмеження узгодженості даних. Модель даних акцентує увагу на тому, які дані потрібні і як вони повинні бути організовані, а не на тому, які операції будуть виконуватися над даними. Модель даних подібна до плану архітектора, який допомагає будувати концептуальні моделі та встановлювати взаємозв'язки між елементами даних.

Існує два типи технік моделювання даних:

- Модель "сутність-зв'язок" (E-R);
- UML (уніфікована мова моделювання).

Основними цілями використання моделі даних є:

- забезпечення точного представлення всіх об'єктів даних, необхідних для бази даних. Відсутність даних призведе до створення помилкових звітів та отримання невірних результатів;
- модель даних допомагає проектувати базу даних на концептуальному, фізичному та логічному рівнях;
- структура моделі даних допомагає визначити реляційні таблиці, первинні та зовнішні ключі і збережені процедури;
- вона дає чітке уявлення про дані бази і може бути використана розробниками баз даних для створення фізичної бази даних;

- вона також корисна для виявлення відсутніх та надлишкових даних;
- хоча початкове створення моделі даних є трудомістким і займає багато часу, в довгостроковій перспективі це робить оновлення та обслуговування IT-інфраструктури дешевшим і швидшим.

Існує три різних типи моделей даних: концептуальні моделі даних, логічні моделі даних та фізичні моделі даних, і кожна з них має певне призначення:

1. концептуальна модель даних. Ця модель даних визначає, що містить система. Ця модель, як правило, створюється зацікавленими сторонами бізнесу та архітекторами даних. Мета – організувати, охопити та визначити бізнес-концепції та правила;
2. логічна модель данихю Визначає як система повинна бути реалізована незалежно від СУБД. Ця модель зазвичай створюється архітекторами даних та бізнес-аналітиками. Метою є розробка технічної карти правил і структур даних4
3. фізична модель данихю Ця модель даних описує, ЯК система буде реалізована з використанням конкретної системи СУБД. Ця модель зазвичай створюється адміністратором баз даних та розробниками. Мета – фактична реалізація бази даних.

2.2.1 Створення концептуальної моделі

Концептуальна модель даних – це представлення концепцій бази даних та їх взаємозв'язків. Метою створення концептуальної моделі даних є встановлення сутностей, їх атрибутів та взаємозв'язків. На цьому рівні моделювання даних навряд чи є якась детальна інформація про фактичну структуру бази даних.

Концептуальна модель даних складається з 3 основних елементів:

- Сутність: Об'єкт реального світу

- Атрибут: Характеристики або властивості сутності
- Зв'язок: Залежність або зв'язок між двома сутностями

Концептуальна модель розробляється незалежно від апаратних специфікацій, таких як ємність сховища даних, місце розташування або специфікацій програмного забезпечення, таких як постачальник і технологія СУБД. Основна увага приділяється представленню даних так, як користувач побачить їх у "реальному світі".

Базові сутності предметної області

1. Сервіс.
2. Сталий тариф.
3. Гнучкий тариф.
4. Тариф трафіку.

Атрибути сутностей:

1. Сервіс:
   - ID;
   - Назва;
   - Тип послуги.
2. Сталий тариф:
   - ID;
   - Сервіс;
   - ЦП;
   - ОЗП;
   - Доступний трафік;
   - Вартість.
3. Гнучкй тариф:
   - ID;
   - Сервіс;
   - ЦП;
   - ОЗП;

- Доступний трафік.

4. Тариф трафіку:
   - ID;
   - Сервіс;
   - Вартість.

2.2.2 Створення логічної моделі

Логічна модель даних використовується для визначення структури елементів даних та встановлення зв'язків між ними. Логічна модель даних додає інформацію до елементів концептуальної моделі даних. Перевагою використання логічної моделі даних є забезпечення фундаменту для формування бази для фізичної моделі. Однак структура моделювання залишається узагальненою (рис. 2.1).

На цьому рівні моделювання даних не визначається первинний або вторинний ключ, необхідно перевірити та скоригувати деталі зв’язків, які були встановлені раніше.

Характеристики логічної моделі даних:

- описує потреби в даних для одного проекту, але може інтегруватися з іншими логічними моделями даних, виходячи за обсяг проекту;
- проектується та розробляється незалежно від СУБД;

– атрибути даних матимуть точні типи та довжину даних.

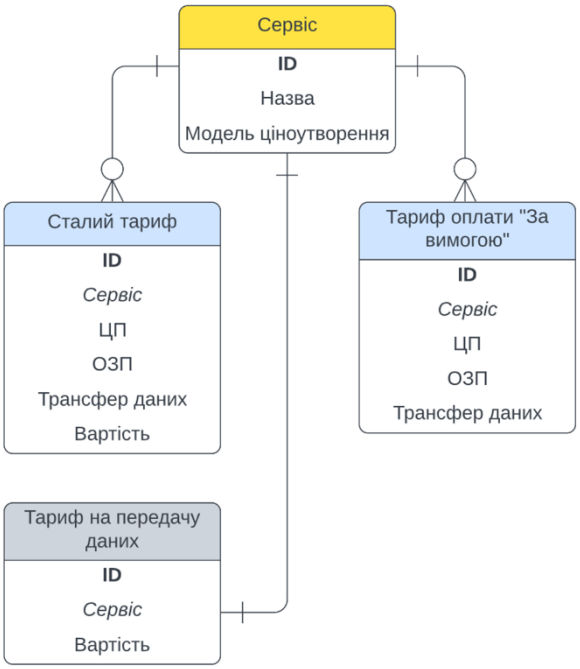

Рисунок 2.1 – Логічна модель

2.2.3 Створення фізичної моделі

Фізична модель даних описує реалізацію моделі даних для конкретної бази даних. Вона пропонує абстракцію бази даних і допомагає генерувати схему. Це пов'язано з кількістю метаданих, які пропонує фізична модель. Фізична модель даних також допомагає у візуалізації структури бази даних шляхом реплікації ключів стовпців бази даних, обмежень, індексів, тригерів та інших функцій СУБД.

Характеристики фізичної моделі даних:

– фізична модель описує потребу в даних для окремого проекту або програми, хоча вона може бути інтегрована з іншими фізичними моделями даних в залежності від обсягу проекту;

– розробляється для конкретної версії СУБД, місця розташування, сховища даних або технології, яка буде використовуватися в проекті;

– стовпці повинні мати точні типи даних, задані довжини та значення за замовчуванням;
– визначаються первинні та зовнішні ключі, представлення, індекси, профілі доступу та повноваження тощо.

Нижче буде приведено фізичну схему таблиць БД.

Таблиця 2.1 – Фізична схема таблиці «Сервіс»

Атрибут	**Властивість**
ID	Int, AI, PK
Name	Varchar(45), Unique
Model	Varchar(45)

Таблиця 2.2 – Фізична схема таблиці «Сталий тариф»

Атрибут	**Властивість**
ID	Int, AI, PK
Service_ID	Int, FK
Cpu	Int
Ram	Int
Data_transfer	Int
Price	Int

Таблиця 2.3 – Фізична схема таблиці «Гнучкий тариф»

Атрибут	**Властивість**
ID	Int, AI, PK
Service_ID	Int, FK

Cpu	Int
Ram	Int
Traffic	Int

Таблиця 2.4 – Фізична схема таблиці «Тариф трафіку»

Атрибут	**Властивість**
ID	Int, AI, PK
Service_ID	Int, FK
Price	Int

Також можна побудувати ER-діаграму (рис. 2.2).

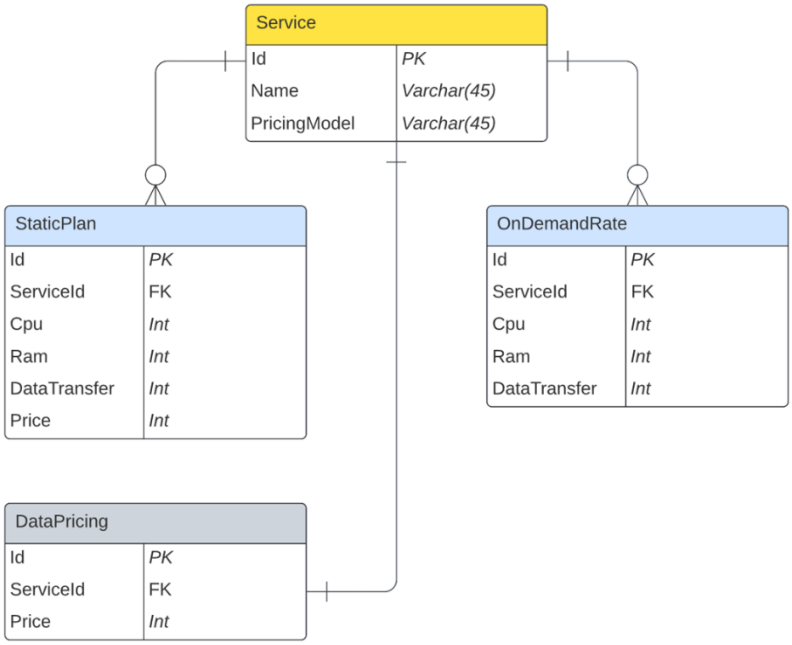

Рисунок 2.2 – ER-діаграма

2.3 Реалізація програмного забезпечення для оцінки вартості розгортання програмного додатку у хмарному середовищі

2.3.1 Створення таблиць

Процес створення таблиць за допомогою мови SQL включає наступні дії:

1. Задання ім’я таблиці.
2. Перелік стовпчиків із зазначенням типів даних атрибутів.
3. Обмеження для кожного стовпчика:
   - приналежність первинному ключу;
   - допустимість невизначених значень;
   - деякі обмеження на значення стовпчика;
   - задання значення за замовчуванням – те, що буде введено, якщо в операторі введення відсутнє конкретне значення для даного поля;
   - приналежність для зовнішнього ключа.

Враховуючи попередні умови, створимо таблиці для бази. Нижче наведений приклад створення однієї таблиці.

CREATE TABLE `static_plan` (
`id` ***int NOT NULL AUTO_INCREMENT***,
`service_id` ***int NOT NULL***,
`cpu` ***int NOT NULL***,
`ram` ***int NOT NULL***,
`traffic` ***int NOT NULL***,
`price` ***int NOT NULL***,
PRIMARY KEY (`id`),
KEY `fk_service_id_idx` (`service_id`),
CONSTRAINT `fk_service_id` ***FOREIGN KEY*** (`service_id`) ***REFERENCES*** `service` (`id`)

2.3.2 Створення індексів

Після створення таблиць слід з’єднати їх, задавши зв’язок ключами. Також створимо індекси для ключів та полів таблиць, до яких найчастіше будуть відбуватися звернення запитів.

2.3.3 Заповнення таблиць даними

Після створення таблиць, первинних і вторинних ключів та індексів необхідно додати дані у базу даних. Даних у таблицях має бути достатньо для подальшої обробки, запитів і звітів, а також аналізу продуктивності інформаційної системи. Є три способи додавання даних у таблиці:

1. Вносити дані використовуючи інтерфейс MySQL Workbench (рис. 2.3).

id	name	model
1	AWS App Runner	flexible
2	GCP App Engine	flexible
3	AWS EC2	standart
4	Azure App Service	standart
5	Alibaba Simple Ap…	standart
NULL	NULL	NULL

Рисунок 2.3 – Внесення даних у таблицю «Service»

2. Вносити дані прописавши SQL команди (INSERT INTO).
3. Вносити дані з таблиць CSV, XLS

2.3.4 Створення запитів та звітів

Для реалізації інформаційної технології необхідні наступні запити та звіти:

1. Вивести доступні тарифні плани сталого типу, які задовольняють введені користувачем критерії, та порахувати вартість.

```
SELECT
service.name as 'name', service.model as 'model' , static_plan.cpu as 'cpu', ROUND(static_plan.ram/1000,2) as 'ram',
ROUND(static_plan.price/1000 + (GREATEST(%(traffic)s - static_plan.traffic, 0))*traffic_pricing.price/1000, 2) as 'total_cost'
FROM service
join static_plan on service.id = static_plan.service_id
join traffic_pricing on service.id = traffic_pricing.service_id
where static_plan.cpu >= %(cpu)s and static_plan.ram/1000 >= %(ram)s
order by total_cost \
LIMIT 5;
```

На рисунку 2.4 зображено вивід результату запиту з такими критеріями: 2 ЦП, 1гб ОЗП, 100гб трафіку.

name	model	cpu	ram	total_cost
AWS EC2	standart	2	1.00	12.87
Alibaba Simple Ap…	standart	2	2.00	15.00
AWS EC2	standart	2	2.00	16.67
Alibaba Simple Ap…	standart	2	4.00	19.00
Azure App Service	standart	2	3.50	26.55

Рисунок 2.4 – Вивід результату виконання запиту

2. Вивести доступні тарифні плани гнучкого типу, які задовольняють введеним користувачем критеріям, та порахувати вартість.

```
SELECT service.name as 'name', service.model as 'model',
```

```
(ROUND((%(cpu)s*%(hours)s*flexible_plan.cpu + %(ram)s*%(hours)s*flexible_plan.ram + %(traffic)s*flexible_plan.traffic)/1000, 2)) as 'total cost'
FROM service join flexible_plan on service.id = flexible_plan.service_id
LIMIT 5;
```

На рисунку 2.5 зображено вивід результату запиту з такими критеріями: 1 ЦП, 2гб ОЗП, 50 гб трафіку.

name	model	total cost
AWS App Runner	flexible	60.66
GCP App Engine	flexible	64.32

Рисунок 2.5 – Вивід результату виконання запиту

3. Вивести доступні тарифні плани гнучкого типу з урахуванням пікових навантажень, які задовольняють введеним користувачем критеріям, та порахувати вартість.

```
SELECT service.name as 'name', service.model as 'model',
(ROUND((%(cpu)s*%(cpu_h)s*flexible_plan.cpu + %(ram_h)s*flexible_plan.ram + %(traffic)s*flexible_plan.traffic)/1000, 2)) as 'total cost'
FROM service join flexible_plan on service.id = flexible_plan.service_id
LIMIT 5;
```

На рисунку 2.6 зображено вивід результату запиту з такими критеріями: 1 ЦП, 2гб ОЗП, пропускна спроможність 50 запитів/с, 8 годин пікового навантаження, 150 запитів/с пікового навантаження, 20 запитів/с звичайне навантаження, 100 гб трафіку.

name	model	total cost
AWS App Runner	flexible	102.60
GCP App Engine	flexible	109.20

Рисунок 2.6 – Вивід результату виконання запиту

4. Вивести доступні тарифні плани сталого та гнучкого типів, з урахуванням пікових навантажень, які задовольняють введеним користувачем критеріям, та порахувати вартість.

```
SELECT service.name as 'name', service.model as 'model', static_plan.cpu as 'cpu', ROUND(static_plan.ram/1000,2) as 'ram',
(case
when service.model = 'flexible' then
	(ROUND((%(cpu)s*%(cpu_h)s*flexible_plan.cpu + %(ram_h)s*flexible_plan.ram + %(traffic)s*flexible_plan.traffic)/1000, 2))
else
	ROUND(
	(case
	when CEILING(%(cpu)s*%(peak)s/static_plan.cpu) > CEILING(%(ram)s*%(peak)s/FLOOR(static_plan.ram/1000))
	then CEILING(%(cpu)s*%(peak)s/static_plan.cpu)
	else
		CEILING(%(ram)s*%(peak)s/FLOOR(static_plan.ram/1000))
	end)
	*static_plan.price/1000 + (GREATEST(%(traffic)s - static_plan.traffic, 0))
	*traffic_pricing.price/1000, 2)
end) as 'total_price'
from service
left join flexible_plan on service.id = flexible_plan.service_id
left join static_plan on service.id = static_plan.service_id
left join traffic_pricing on service.id = traffic_pricing.service_id
where (static_plan.cpu >= %(cpu)s and static_plan.ram >= %(ram)s) or (static_plan.cpu is null and static_plan.ram is null)
```

```
order by total_price
LIMIT 5;
```

На рисунку 2.7 зображено вивід результату запиту з такими критеріями: 1 ЦП, 2гб ОЗП, пропускна спроможність 50 запитів/с, 8 годин пікового навантаження, 150 запитів/с пікового навантаження, 20 запитів/с звичайне навантаження, 100 гб трафіку.

name	model	cpu	ram	total_price
Alibaba Simple Ap...	standart	1	1.00	27.00
AWS EC2	standart	2	2.00	32.01
AWS EC2	standart	2	1.00	32.22
Alibaba Simple Ap...	standart	2	4.00	38.00
Alibaba Simple Ap...	standart	2	2.00	45.00

Рисунок 2.7 – Вивід результату виконання запиту

Висновки до розділу 2

У розділі визначено підходи та засоби для реалізації інформаційної технології, зокрема для зберігання даних вибрано реляційну СУБД MySQL. За результатами дослідження предметної області розроблено діаграму «сутність-зв’язок», на основі якої створено логічну та фізичну моделі даних БД.

На основі розробленої фізичної моделі створено базу даних, основними таблицями якої є таблиці «Сервіс», «Сталий тариф», «Гнучкий тариф» та «Тариф трафіку». Таблиці були заповнені даними, які відображають інформацію про ресурси хмарних середовищ AWS App Runner, GCP App Engine, AWS EC2, Azure App Service, Alibaba Simple Application Server та їх тарифи на ресурси.

Для реалізації інформаційної технології також створено ряд необхідних запитів, які дозволяють отримати дані із бази даних.

3. РОЗРОБКА ДОДАТКУ АГРЕГАТОРА СЕРВІСІВ

3.1 Вибір засобів реалізації додатку

Веб-фреймворк – це програмна платформа, яка була розроблена з метою спрощення процесу веб-розробки та полегшення створення веб-сайтів. Він включає в себе можливості шаблонування, які дозволяють представляти інформацію в браузері, надає середовище для написання сценаріїв потоків інформації, а також містить багато інтерфейсів прикладного програмування (API) для отримання доступу до базових ресурсів даних. Більшість фреймворків також надають інструменти для того, щоб веб-розробники могли створити систему управління контентом (CMS) для управління цифровою інформацією на веб-сайтах та в Інтернеті.

Веб-додаток або фреймворк розробки можна розглядати як заздалегідь побудовану структуру, яка обробляє більш повторювані процеси та функції, пов'язані з розробкою веб-сайту. Це означає, що веб-розробник витрачає більшу частину свого часу на взаємодію з різними частинами веб-фреймворку за допомогою коду.

Деякі з функцій, пов'язаних з веб-фреймворками, включають веб-кешування, процедури аутентифікації та авторизації, відображення та конфігурацію баз даних та відображення URL-адрес.

Найбільш помітні переваги для веб-розробників, які використовують веб-фреймворки, пов'язані з тим, що вони мають відкритий код, ефективні, мають високий рівень підтримки, мають високий рівень безпеки і включають в себе функцію інтеграції.

Відкритий код означає, що веб-фреймворки є дуже економічно ефективними як для розробника, так і для клієнта. Це не означає, що вони не є якісними.

Більшість популярних веб-фреймворків, що використовуються розробниками, є безкоштовними для використання.

Ефективність – ще одна важлива перевага для веб-розробників. Це пов'язано з тим, що веб-фреймворки усувають необхідність написання великої кількості повторюваного коду, що дозволяє розробникам створювати веб-сайти та додатки набагато швидше. Веб-фреймворк також поставляється з командою підтримки та розширеними функціями безпеки, що означає, що можна бути впевненим, знаючи, що якщо проблема все ж виникне, вона буде вирішена командою експертів.

Однією з найбільш корисних функцій веб-фреймворку є функція інтеграції, яка дозволяє розробникам пов'язувати інші інструменти, такі як бази даних, з фреймворком.

Однак, розробники часто відзначають, що найбільші недоліки використання веб-фреймворків пов'язані з тим, що вони дуже обмежені в плані внесення змін до веб-фреймворку. Все, починаючи від парадигм кодування і закінчуючи дизайном, є дуже обмежувальним. Іншим недоліком є те, що деяким розробникам важко вивчити мову, яка лежить в основі веб-фреймворків. Часто веб-розробники вивчають фреймворк, але не знають мови кодування, що лежить в його основі, що ускладнює для них повне розуміння всього, що стосується самого фреймворку.

Для розробки буде використана мова програмування Python, отже треба вибрати фреймворк на Python.

На сьогоднішній день Python це найпопулярніша мова програмування, і однією з вагомих причин для цього є його чудова колекція фреймворків та бібліотек. Python має, мабуть, найпотужніші, найактивніші та найпродуктивніші фреймворки, коли мова йде про веб-розробку. Найпопулярнішим серед них, за результатами останнього опитування порталу JetBrains, став Flask (рис. 3.1).

Flask – це фреймворк Python, доступний під ліцензією BSD. Він був натхненний фреймворком Sinatra Ruby. Flask залежить від інструментарію Werkzeug WSGI та шаблонізатору Jinja2.

Основна ідея Flask полягає в тому, щоб допомогти побудувати міцний фундамент для веб-додатків. Звідти вже можна використовувати будь-які

розширення, які можуть знадобитися. Flask обирають для будь-яких проектів. Фактично, це вибір за замовчуванням для будь-якого веб-проекту, який не підходить для Django.

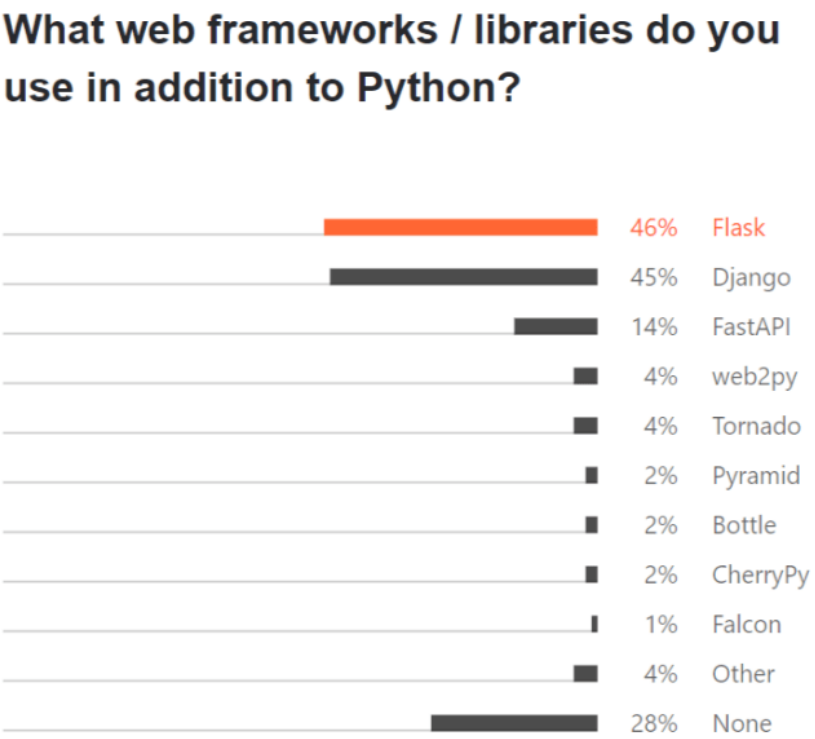

Рисунок 3.1 – Результати опитування порталу JetBrains

Легкий та модульний дизайн Flask дозволяє легко адаптувати його до потреб розробників. Він включає в себе ряд корисних функцій "з коробки":

- вбудований сервер розробки та швидкий відладчик;
- інтегрована підтримка модульного тестування;
- RESTful диспетчеризація запитів;
- шаблонування Jinja2;
- підтримка безпечних файлів cookie (сесії на стороні клієнта);
- відповідність WSGI 1.0;
- можливість підключення будь-якого ORM;
- обробка HTTP запитів.

З моменту запуску в 2010 році Flask був оновлений 27 разів. На сьогоднішній день він залишається найбільш швидкозростаючим фреймворком Python.

Django – це безкоштовний фулстек фреймворк Python з відкритим вихідним кодом. Він намагається включити всі необхідні функції за замовчуванням, а не пропонувати їх як окремі бібліотеки.

Деякі зі зразкових функцій Django включають автентифікацію, маршрутизацію URL-адрес, механізм шаблонів, об'єктно-реляційний маппер (ORM) та міграцію схем баз даних (Django v.1.7+).

Ці функції роблять Django дуже масштабованим, дуже швидким і надзвичайно універсальним.

Django використовує свій ORM для зіставлення об'єктів з таблицями баз даних. Один і той же код працює з різними базами даних і його не важко перенести з однієї бази даних в іншу. Основними базами даних, з якими працює Django, є PostgreSQL, MySQL, SQLite та Oracle, але сторонні драйвери дозволяють використовувати й інші.

За допомогою Django можна створювати будь-які веб-додатки - від невеликих проектів до складних веб-сайтів. Завдяки своїй гнучкості Django також використовується для створення MVP, що дозволяє стартапам оптимізувати свій час та бюджет.

Pyramid – це фреймворк веб-додатків з відкритим вихідним кодом на основі Python. Його мета - зробити якомога більше з мінімальною складністю. Працюючи на Python 3, Pyramid йде в ногу з технологічними вдосконаленнями.

Найяскравішою особливістю Pyramid є її здатність добре працювати як з малими, так і з великими додатками. Деякі з чудових можливостей Pyramid включають:

- однофайлові додатки;
- генерація URL-адрес;
- розширювана конфігурація;
- всеохоплюючі шаблони та специфікації ресурсів;
- гнучка аутентифікація та авторизація;

– тестування, підтримка та вичерпна документація.

Серед веб-фреймворків на Python, Flask підходить для інформаційної технології найкраще.

3.2 Проектування додатку агрегатора сервісів

Реалізація додатку агрегатору сервісів починається з вирішення, з яких сторінок він буде складатися.

Сторінки агрегатора:

– Домашня сторінка;
– Сторінка сталих тарифів;
– Сторінка гнучких тарифів;
– Сторінка гнучких тарифів з урахуванням пікових навантажень;
– Сторінка порівняння сталих і гнучких тарифів.

Як зазначено у розділі 3.1, у розробці буде використано фреймворк Flask мови програмування Python. Додатково буде використано бібліотеки SQLAlchemy, WTForms та рушій шаблонів Jinja.

Jinja – швидкий, розширюваний рушій шаблонів. Спеціальні заповнювачі дозволяють писати код в шаблоні, подібний до синтаксису Python. Потім шаблону передаються дані для рендерингу кінцевого документа. Включає в себе:

– включення та спадкування шаблонів;
– визначення та імпорт макросів всередині шаблонів;
– ізольоване середовище дозволяє безпечно рендерити ненадійні шаблони;
– підтримка AsyncIO для генерації шаблонів і виклику асинхронних функцій;
– шаблони компілюються в оптимізований код Python і кешуються, або можуть бути скомпільовані заздалегідь;
– розширювані фільтри, тести, функції і навіть синтаксис.

Філософія Jinja полягає в тому, що хоча логіка програми повинна бути написана на Python, якщо це можливо, вона не повинна ускладнювати роботу дизайнера шаблонів, занадто сильно обмежуючи функціональність.

SQLAlchemy – це бібліотека для роботи з реляційними СУБД із застосуванням технології ORM. Служить для синхронізації об'єктів Python і записів реляційної бази даних. SQLAlchemy дає змогу описувати структури баз даних і способи взаємодії з ними мовою Python без використання SQL.

Використання SQLAlchemy для автоматичної генерації SQL-коду має кілька переваг порівняно зі звичайним написанням SQL:

- безпека. Параметри запитів екрануються, що робить атаки типу SQL injection малоймовірними;
- продуктивність. Для виконання запитів створюється план виконання, який може бути використаний для повторних виконань;
- переносимість. SQLAlchemy дає змогу писати код на Python, сумісний із кількома back-end СУБД. Незважаючи на стандартизацію мови SQL, між базами даних є відмінності в її реалізації, абстрагуватися від яких і допомагає SQLAlchemy.

WTForms – це гнучка бібліотека валідації та рендерингу форм для веб-розробки на Python. Вона може працювати з будь-яким веб-фреймворком та шаблонізатором. Вона підтримує валідацію даних, захист CSRF, інтернаціоналізацію (I18N) та багато іншого.

Всього в агрегаторі буде присутньо 4 форми: форма для сталих тарифів, форма для гнучких тарифів, форма для гнучких тарифів з врахуванням пікових навантажень та форма для сторінки порівняння сталих та гнучких тарифів.

Форма для сталих тарифів буде складатися з наступних полів:

- Поле кількості ЦП;
- Поле кількості ОЗП;
- Поле кількості трафіку.

Форма для гнучких тарифів буде складатися з:

- Поле кількості ЦП;

– Поле кількості ОЗП;
– Поле кількості робочих годин;
– Поле кількості трафіку.

Форма для гнучких тарифів з врахуванням пікових навантажень складається з:

– Поле кількості ЦП;
– Поле кількості ОЗП;
– Поле пропускної спроможності;
– Поле годин пікових навантажень;
– Поле пікової кількості запитів на секунду;
– Поле звичайної кількості запитів на секунду;
– Поле кількості трафіку.

Форма для сталих та гнучких тарифів з врахуванням пікових навантажень складається з наступних полів:

– Поле кількості ЦП;
– Поле кількості ОЗП;
– Поле пропускної спроможності;
– Поле годин пікових навантажень;
– Поле пікової кількості запитів на секунду;
– Поле звичайної кількості запитів на секунду;
– Поле кількості трафіку.

3.3 Реалізація додатку агрегатора сервісів

3.3.1 Структура файлів додатку

Створення веб-додатку у фреймворку Flask розпочинається із налаштування файлу ініціалізації «__init__.py», також тут буде підключено бібліотеки.

У файлі «models.py» будуть створені класи, які відповідають таблицям бази даних, та відношення «один-до-одного», «один-до-багатьох» які будуть використовуватись SQLAlchemy, а також функції для запитів до бази даних.

Класи для роботи з формами винесені у файл «forms.py».

У папці “templates” будуть знаходитись шаблони сторінок.

Для файлів стилів та зображень створена окрема папка “static”.

3.3.2 Реалізація моделей для роботи з базою даних

Моделі, які будуть використовуватися SQLAlchemy для роботи з базою даних мають відповідати таблицям, створеними у розділі 2.

Для прикладу розглянемо модель, відповідну до таблиці сервісів, вона має наступний вигляд:

```
class Service(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(45), unique=True, nullable=False)
    model = db.Column(db.String(45), nullable=False)
    service_plans = db.relationship(
        'StandartPlan', backref='service', lazy='dynamic')
```

Тут вказані поля таблиці, типи даних, допустимі значення, ключі, відношення.

3.3.3 Обробка форм

Для роботи з формами створюються відповідні класи. Для прикладу розглянемо форму, яка буде використовуватись на сторінці розрахунків сталих та гнучких тарифів з врахуванням пікових навантажень:

```
class StaticFlexibleForm(FlaskForm):
    cpu = IntegerField('CPU')
    ram = IntegerField('RAM')
    conc = IntegerField('Concurrency')
    peak_h = IntegerField('Peak hours')
    peak_rps = IntegerField('Peak requsts/second')
    norm_rps = IntegerField('Normal requsts/second')
    traffic = IntegerField('Traffic')
    submit = SubmitField('Submit')
```

Валідація виглядає наступним чином:

```
@app.route("/static_and_flexible", methods=['GET', 'POST'])
def static_and_flexible():
    form = StaticFlexibleForm()
    if form.validate_on_submit():
        …
```

3.3.4 Створення сторінок

На домашній сторінці будуть описані доступні у додатку функції та відповідні посилання (рис. 3.1). Приклад посилання з використанням Jinja:

```
<a class="a_no_hover" href="{{ url_for('flexible_plan') }}"> Гнучкі тарифи </a>
```

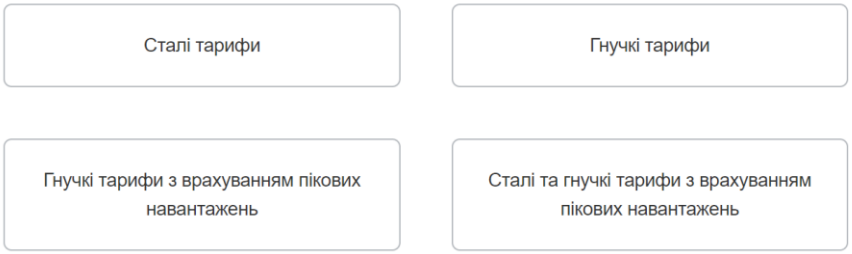

Рисунок 3.1 – Домашня сторінка

Сторінки з розрахунками виглядають схожим чином, тому, для прикладу розглянемо сторінку для розрахунків сталих тарифів. Логіка описана у файлі маршрутизації:

```
@app.route("/static_plan", methods=['GET', 'POST'])
def static_plan():
    form = StaticForm()
    if form.validate_on_submit():
        query = estimate_static_plan(form.cpu.data, form.ram.data, form.traffic.data)
        return render_template('static_plan.html', form=form, q=query)
    return render_template('static_plan.html', form=form)
```

Було дозволено метод HTTP POST для отримання даних від користувача, проініціалізовано форму. Користувачу повертається сторінка з формою (рис. 3.2).

Рисунок 3.2 – Форма на сторінці сталих тарифів

Якщо валідація даних, отриманих від користувача, відбулася успішно, відбувається запит у базу даних та отриманий результат повертається користувачу (рис. 3.3).

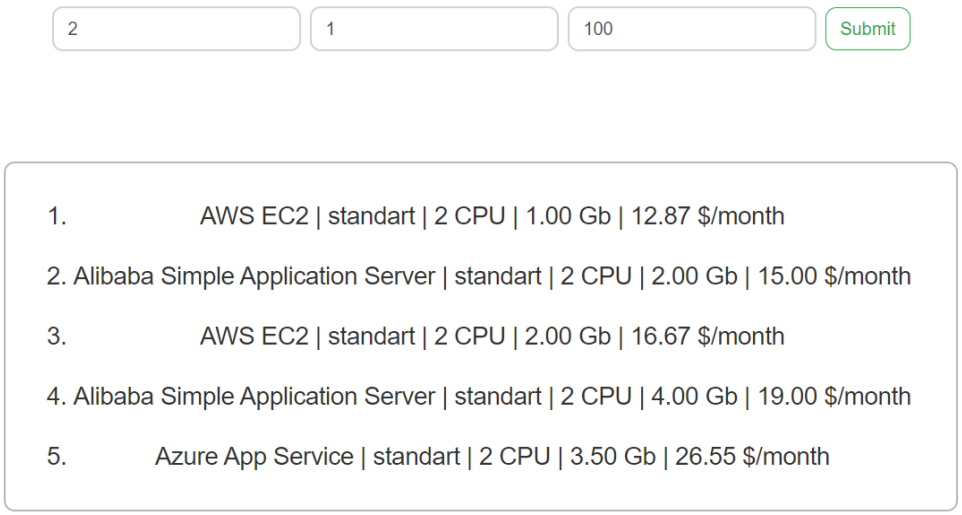

Рисунок 3.3 – Результат обчислень на сторінці сталих тарифів

Для розгортання додатку знадобиться пристрій на базі операційної системи Windows, Linux або MacOS, встановлені MySQL, Python 3.x.

Встановлюються фреймворки та бібліотеки для Python з файлу «requirements.txt» командою:

pip install -r /path/to/requirements.txt

або

pip3 install -r /path/to/requirements.txt

Розгортається додаток виконанням у корневій папці додатку команди:

Python run.py

За замовчуванням, додаток розгортається за адресою http://127.0.0.1:5000/.

3.4 Тестування агрегатора сервісів

Для перевірки результатів обчислень, порівняємо отримані результати з оцінками нативних калькуляторів сервісів.

Для прикладу, порівняємо розрахунки оцінки вартості використання гнучкого тарифу розробленого додатку та калькулятора AWS. Дані для перевірки: 1 ЦП, 2 гб ОЗП, пропускна здатність 50 запитів у секунду, 8 годин на день пікового трафіку, 200 запитів у пікових періодах, та 30 у нормальному режимі.

У результаті обчислень, дані, отримані у розробленому додатку (рис. 3.4) та оцінка нативного калькулятора (рис. 3.5) відрізняються менше ніж на 0.1%.

Інформаційна технологія точно оцінює додані на даному етапі розробки, та пропонує комбінацію сервіса та типу послуги найменшої вартості

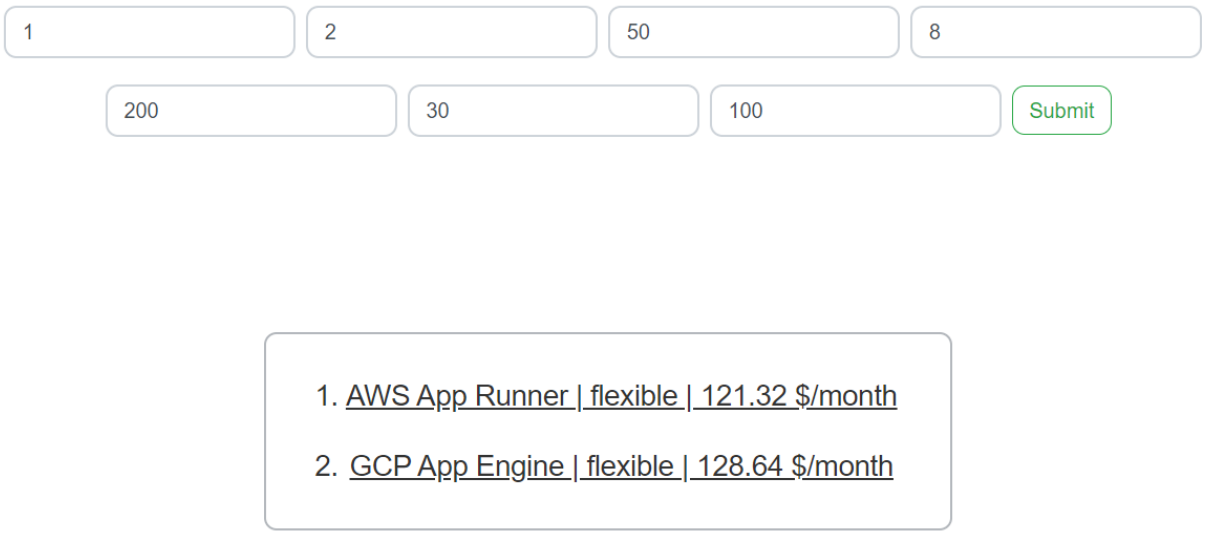

Рисунок 3.4 = Результат розрахунків розробленого агрегатора

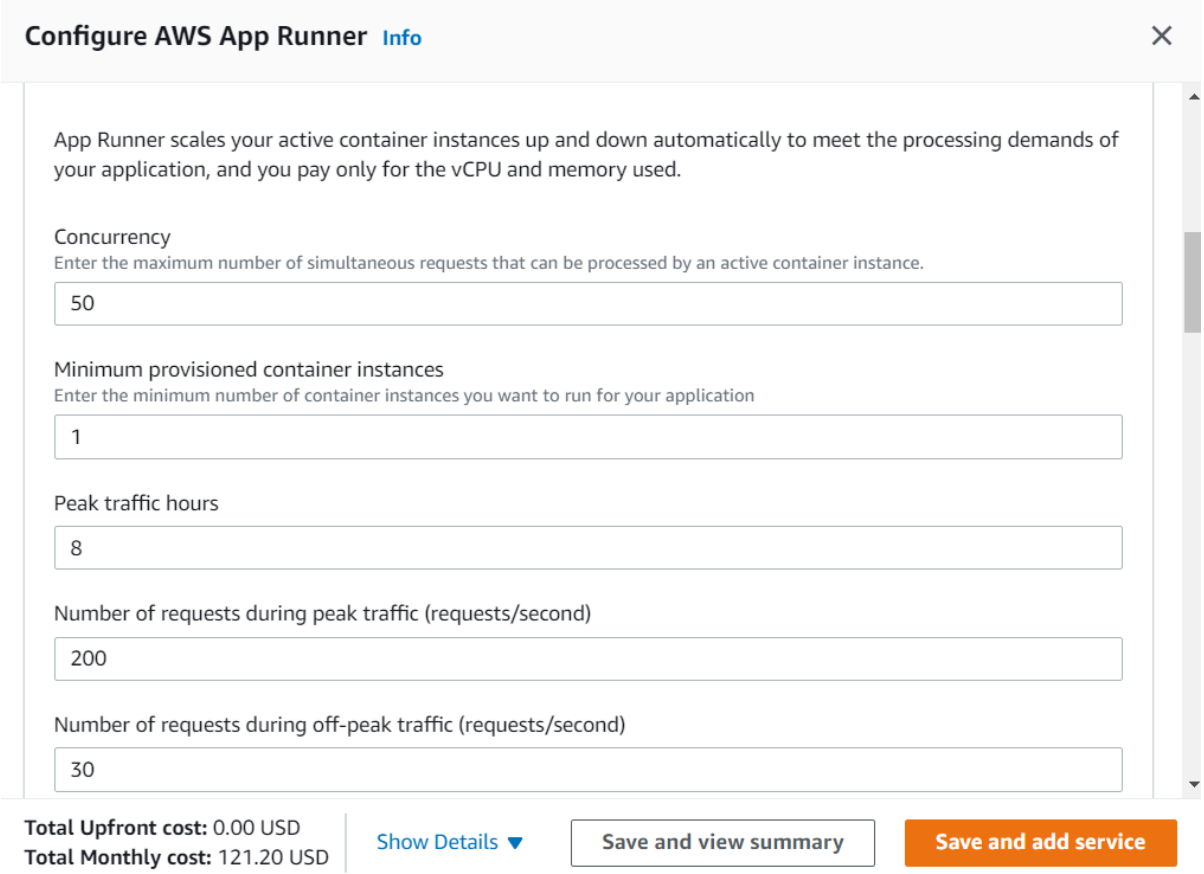

Рисунок 3.5 – Оцінка вартості у калькуляторі AWS

Висновки до розділу 3

У розділі представлено проектування та реалізацію агрегатора сервісів, який дозволяє користувачу задати парметри розгортання додатку та обчислює вартість розгортання для різних хмарних середовищ. Для реалізації агрегатора сервісів використано мову програмування Python, фреймворк Flask, бібліотеки SQLAlchemy, WTForms та рушій шаблонів Jinja.

Тестування агрегатора проводилось порівнянням розрахунків оцінки вартості використання гнучкого тарифу на калькуляторі хмарного провайдерат AWS. У результаті обчислень, дані, отримані у розробленому додатку та оцінка нативного калькулятора відрізняються менше ніж на 0.1%.

ВИСНОВОК

У кваліфікаційній роботі було проаналізовано процес розгортання програмного забезпечення на прикладі розгортання вебдодатку на обчислювальному ресурсі хмарного середовища. Також описано основні переваги та недоліки хмарних обчислень та розглянуто їх існуючі моделі: інфраструктура як послуга, платформа як послуга та програмне забезпечення як послуга. Особливу увагу приділену сутності "інстанс", його ролі у хмарних обчисленнях, типам, життєвому цикл. Також у роботі розглянуто послуги провайдерів хмарних технологій у контексті використання інстансів як обчислювального ресурсу, проаналізовано проблеми оцінки вартості розгортання додатку у хмарному середовищі, фактори, які впливають на вартість, існуючі типи витрат, моделі ціноутворення.

На основі аналізу предметної області було створено ER-модель, визначено базові сутності та атрибути сутностей предметної області. Між сутностями визначені і встановлені типи відносин. На основі спроектованої ER-діаграми створені логічна та фізична моделі бази даних, а також створена та заповнена данимибаза даних, яка є невідємною складовою інформаційної системи. База даних створена в реляційній системі керування базами даних MySQL.

Розроблено агрегатор сервісів провайдерів хмарних ресурсів з використанням фреймворку Flask мови програмування Python. Також дляйого реалізації були використані бібліотеки SQLAlchemy, WTForms та рушій шаблонів Jinja.

Тестування агрегатора проводилось порівнянням розрахунків оцінки вартості використання гнучкого тарифу на калькуляторі хмарного провайдерат AWS. У результаті обчислень, дані, отримані у розробленому додатку та оцінка нативного калькулятора відрізняються менше ніж на 0.1%.

Інформаційна технологія оцінює додані на даному етапі розробки сервіси та пропонує комбінацію сервіса та типу послуги найменшої вартості, а отже, може бути використана організаціями та розробниками в оцінці вартості розгортання додатку у хмарному середовищі та допоможе зменшити витрати.

ПЕРЕЛІК ПОСИЛАНЬ

1. C. Qu, R. N. Calheiros, R. Buyya, Auto-scaling web applications in clouds: A taxonomy and survey, ACM Comput. Surv. 51 (4) (2018) 73:1–73:33.
2. K. Hightower, B. Burns, J. Beda, Kubernetes: Up and Running Dive into the Future of Infrastructure, 1st Edition, O'Reilly Media, Inc., 2017.
3. Google App Engine, https://developers.google.com/appengine/ (дата звернення 11.10.2022).
4. Microsoft Azure, http://azure.microsoft.com (дата звернення 11.10.2022).
5. DevOps, http://devops.com (дата звернення 11.10.2022).
6. Kubernetes Authors, Kubernetes, https://kubernetes.io (дата звернення 11.10.2022).
7. Amazon Simple Storage Service (S3). http://www.amazon.com/s3/ (дата звернення 11.10.2022).
8. R. D. Cosmo, J. Mauro, S. Zacchiroli, G. Zavattaro, Aeolus: A component model for the cloud, Inf. Comput. 239 (2014).
9. M. Burgess, A Site Configuration Engine, Computing Systems.
10. J. Z. Li, C. M. Woodside, J. W. Chinneck, M. Litoiu, Cloudopt: Multi-goal optimization of application deployments across a cloud, in: CNSM, IEEE, 2011, C. 1–9.
11. W. Li, P. Sv¨ard, J. Tordsson, E. Elmroth, Cost-optimal cloud service place ment under dynamic pricing schemes, in: IEEE/ACM UCC, IEEE Computer Society, 2013, C. 187–194.
12. S. Gouw, J. Mauro, G. Zavattaro, On the Modeling of Optimal and Automatized Cloud Application Deployment, 2018.
13. O. H. Nejat, H. Motameni, H. V. Nejad, An integrated optimal method for cloud service ranking, 2021.

14. Hussain A, Chun J, Khan M. A novel customer-centric Methodology for Optimal Service Selection (MOSS) in acloud environment. Future Generation Computer Systems 2020; 105:562-580.
15. Iosup A, Ostermann S, Yigitbasi MN, Prodan R, Fahringer T et al. Performance analysis of cloud computingservices for many-tasks scienti□c computing. IEEE Transactions on Parallel and Distributed systems 2011; 22 (6):931-945.
16. Alabool H, Kamil A, Arshad N, Alarabiat D. Cloud service evaluation method-based Multi-Criteria Decision-Making: A systematic literature review. Journal of Systems and Software 2018; 139:161-188.
17. D. Cunha, P. Neves, P. Sousa, PaaS Manager: A Platform-as-a-Service Aggregation Framework, 2014, Computer Science and Information Systems 11(4):1209-1228.
18. T. Zou, R. Le Bras, M. Salles, ClouDiA: A deployment advisor for public clouds, 2015.
19. R. Pal, S. Mishra, P..Patnaik, Study on Cost Estimation of Service Delivery in Cloud Computing Environment, 2014.
20. Saravanan K, and Sri BighnaHema, "Dynamic Pricing Model for a Cloud Cache Environment" International Journal of Engineering Trends and Technology (IJETT) - Volume4Issue4- April 2013.
21. M. Graiet, L. Hamel, A. Mammar, and S. Tata, "A verification and deployment approach for elastic component-based applications," Formal Aspects of Computing, Mar 2017.
22. E. Albert, M. Gómez-Zamalloa, and M. Isabel, "SYCO: A systematic testing tool for concurrent objects," in 25th International Conference on Compiler Construction (CC'16). ACM, 2016, C. 269–270.
23. I. Stupar, D. Huljenić, Model-Based Cloud Service Deployment Optimisation Method for Minimisation of Application Service Operational Cost, 2021.
24. Stupar, I., Huljeni´c, D.: Model-based extraction of knowledge about the effect of cloud application context on application service cost and quality of service. Scientific Programming 2019 (2019).

25. Karim, B., Tan, Q., Villar, J.R., de la Cal, E.: Resource brokerage ontology for vendor-independent cloud service management. In: 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), C. 466–472 (2017). IEEE.
26. Di, S., Robert, Y., Vivien, F., Kondo, D., Wang, C.-L., Cappello, F.: Optimization of cloud task processing with checkpoint-restart mechanism. In: SC' 13, C. 1–12 (2013).
27. Dubois, D.J., Casale, G.: Autonomic provisioning and application mapping on spot cloud resources. In: IEEE ICCAC '15 (2015).
28. Yao, M., Zhang, P., Li, Y., Hu, J., Lin, C., Li, X. Y.: Cutting your cloud computing cost for deadline-constrained batch jobs. In: IEEE ICWS '14, C. 337–344 (2014).
29. Bellur, U., Malani, A., Narendra, N.: Cost optimization in multi-site multi-cloud environments with multiple pricing schemes. In: IEEE CLOUD '14, C. 689–696 (2014).
30. Simonet, A., Lebre, A., Orgerie, A. (2016). Deploying distributed cloud infrastructures: Who and at what cost? In: 2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW), C. 178–183.

ДОДАТОК А

Файл ініціалізації, підключення SQLAlchemy:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SECRET_KEY'] = 'секретний ключ'
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://{ім’я користувача
бази даних}:{пароль}@{адреса розташування бази даних} /{назва бази даних}
db = SQLAlchemy(app)

from project import routes
```

ДОДАТОК Б

Файл «models.py». Моделі SQLAlchemy для роботи з базою даних.

```
from project import db

#клас відповідає таблиці у базі даних
class Service(db.Model):
    #вказуються типи даних, та параметри полів
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(45), unique=True, nullable=False)
    model = db.Column(db.String(45), nullable=False)
    #відношення «один до багатьох»
    plans = db.relationship(
        'StandartPlan', backref='service', lazy='dynamic')

    #оформлення виводу
    def __repr__(self):
        return f"Service('{self.name}')"

class StaticPlan(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    service_id = db.Column(db.Integer, db.ForeignKey('service.id'))
    cpu = db.Column(db.Integer, nullable=False)
    ram = db.Column(db.Integer, nullable=False)
    traffic = db.Column(db.Integer)
    price = db.Column(db.Integer, nullable=False)
```

```
    def __repr__(self):
        return f"Plan('{self.cpu}, {self.ram}, {self.price}')"

class FlexiblePlan(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    service_id = db.Column(db.Integer, db.ForeignKey('service.id'))
    cpu = db.Column(db.Integer, nullable=False)
    memory = db.Column(db.Integer, nullable=False)
    traffic = db.Column(db.Integer, nullable=False)

    def __repr__(self):
        return f"Rates('{self.cpu}, {self.memory}, {self.price}')"

class TrafficPricing(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    service_id = db.Column(db.Integer, db.ForeignKey('service.id'))
    price = db.Column(db.Integer, nullable=False)
```

ДОДАТОК В

Файл «forms.py». Класи для роботи з формами.

```
from flask_wtf import FlaskForm
from wtforms import SubmitField, IntegerField

class StaticForm(FlaskForm):
    #вказується типи даних полів форми
    cpu = IntegerField('CPU')
    ram = IntegerField('RAM')
    traffic = IntegerField('Traffic')
    submit = SubmitField('Submit')

class FlexibleRatesForm(FlaskForm):
    cpu = IntegerField('CPU')
    ram = IntegerField('RAM')
    hours = IntegerField('Hours')
    traffic = IntegerField('Traffic')
    submit = SubmitField('Submit')

class FlexibleScalingForm(FlaskForm):
    cpu = IntegerField('CPU')
    ram = IntegerField('RAM')
    conc = IntegerField('Concurrency')
    peak_h = IntegerField('Peak hours')
    peak_rps = IntegerField('Peak requsts/second')
    norm_rps = IntegerField('Hormal requsts/second')
```

```
    traffic = IntegerField('Traffic')
    submit = SubmitField('Submit')

class StaticFlexibleForm(FlaskForm):
    cpu = IntegerField('CPU')
    ram = IntegerField('RAM')
    conc = IntegerField('Concurrency')
    peak_h = IntegerField('Peak hours')
    peak_rps = IntegerField('Peak requsts/second')
    norm_rps = IntegerField('Hormal requsts/second')
    traffic = IntegerField('Traffic')
    submit = SubmitField('Submit')
```

Підключення форми до сторінки має наступний вигляд:

```
#тут додається токен форми, що робить її безпечною.
{{ form.hidden_tag() }}

#для стилізаці було використано набір інструментів Bootstrap.
<fieldset class="form-group d-inline-flex mb-10">
        <div class="form-group">
                #тут «form» це назва змінної, якій належить форма, «cpu» - назва поля.
                {{ form.cpu(class="form-control mr-sm-2", style="border: 2px solid #ced4da; border-radius: .6rem;", placeholder="CPU") }}
        </div>
</fieldset>
```

ДОДАТОК Г

Файл «routes.py». Маршрутизація.

```
from flask import render_template
from project import app
from project.models import estimate_flexible_plan, estimate_flexible_scaling, estimate_static_flexible, estimate_static_plan
from project.forms import StaticFlexibleForm, StaticForm, FlexibleRatesForm, FlexibleScalingForm

#домашня сторінка
@app.route("/")
@app.route("/home", methods=['GET', 'POST'])
def home():
    return render_template('home.html')

#сторінка сталих тарифів
@app.route("/static_plan", methods=['GET', 'POST'])
def static_plan():
    form = StaticForm()
    if form.validate_on_submit():
        query = estimate_static_plan(form.cpu.data, form.ram.data, form.traffic.data)
        return render_template('static_plan.html', form=form, q=query)
    return render_template('static_plan.html', form=form)

#сторінка гнучких тарифів
@app.route("/flexible_plan", methods=['GET', 'POST'])
```

```
def flexible_plan():
    form = FlexibleRatesForm()
    if form.validate_on_submit():
        query = estimate_flexible_plan(form.cpu.data, form.ram.data, form.hours.data, form.traffic.data)
        return render_template('flexible_plan.html', form=form, q=query)
    return render_template('flexible_plan.html', form=form)

#сторінка гнучких тарифів з урахуванням пікових навантажень
@app.route("/flexible_scaling", methods=['GET', 'POST'])
def flexible_scaling():
    form = FlexibleScalingForm()
    if form.validate_on_submit():
        query = estimate_flexible_scaling(form.cpu.data, form.ram.data, form.conc.data, form.peak_h.data, form.peak_rps.data, form.norm_rps.data, form.traffic.data)
        return render_template('flexible_scaling.html', form=form, q=query)
    return render_template('flexible_scaling.html', form=form)

# сторінка порівняння сталих і гнучких тарифів з урахуванням пікових навантажень
@app.route("/static_and_flexible", methods=['GET', 'POST'])
def static_and_flexible():
    form = StaticFlexibleForm()
    if form.validate_on_submit():
        query = estimate_static_flexible(form.cpu.data, form.ram.data, form.conc.data, form.peak_h.data, form.peak_rps.data, form.norm_rps.data, form.traffic.data)
        return render_template('static_and_flexible.html', form=form, q=query)
    return render_template('static_and_flexible.html', form=form)
```