

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

**Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу**

«До захисту допущено»

Завідувач кафедри

_____ О. Л. Тимошук

« ____ » _____ 2021 р.

**Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи і методи штучного інтелекту»
спеціальності 122 «Комп'ютерні науки»
на тему: «Модель управління ресурсами гетерогенних баз даних в хмарному
середовищі»**

Виконав:

студент IV курсу, групи ІС-зп92
Єрмоленко Станіслав Олександрович

Керівник: професор кафедри ММСА,
д.т.н., професор, Мухін В.Є.

Консультант з економічного розділу:
к.е.н., доцент, Рощина Н.В.

Консультант з нормоконтролю:
к.т.н., доцент, Коваленко А.Є.

Рецензент: професор кафедри ІСТ,
д.т.н., доцент Корнага Я. І.

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без відповідних
посилань.

Студент _____

Київ – 2021

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О. Л. Тимошук

«__» _____ 20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Єрмоленку Станіславу Олександровичу

1. Тема роботи «Модель управління ресурсами гетерогенних баз даних в хмарному середовищі», керівник Мухін Вадим Євгенович, професор кафедри ММСА, затверджені наказом по університету від «__» _____ 20__ р. № _____
2. Термін подання студентом роботи: 22 вересня 2021 року
3. Вихідні дані до роботи: системи управління базами даних хмарних провайдерів
4. Зміст роботи: дослідження систем управління базами даних; розробка власного програмного продукту для імплементації та роботи з базами даних в хмарному середовищі.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація з результатами роботи: модель архітектури СУБД; огляд та аналіз сервісів хмарних провайдерів, обрана модель реалізації та її особливості; схема роботи аплікації.
6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н. В., доцент		

7. Дата видачі завдання 7 лютого 2021 року

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Формулювання тематики дослідження	07.02.21 — 16.02.21	
2.	Аналіз актуальності теми	16.02.21 — 19.02.21	
3.	Формулювання задачі дослідження	20.02.21 — 28.02.21	
4.	Збір інформації та її аналіз	01.03.21 — 06.06.21	
5.	Написання програми	07.06.21 — 04.07.21	
6.	Оформлення пояснювальної записки	05.07.21 — 30.08.21	
7.	Підготовка презентації для захисту	31.08.21 — 20.09.21	
8.	Попередній захист дипломної роботи	20.09.21 — 24.09.21	
9.	Захист дипломної роботи	25.09.21	

Студент Станіслав ЄРМОЛЕНКО

Керівник Вадим МУХІН

РЕФЕРАТ

Дипломна робота: 136 с., 6 табл., 39 рис., 2 додатка, 21 джерело.

ХМАРНЕ СЕРЕДОВИЩЕ, РЕЛЯЦІЙНІ ТА НЕ-РЕЛЯЦІЙНІ БАЗИ ДАНИХ,
МОДЕЛЬ УПРАВЛІННЯ РЕСУРСАМИ, ЗБЕРІГАННЯ ДАНИХ.

Мета дипломної роботи полягає в дослідженні систем управління базами даних, подальшому вибору ефективної моделі управління ресурсами гетерогенних баз даних в хмарному середовищі та здійснення імплементації аплікації з різними видами баз даних та веб-інтерфейсом.

Об'єктом дослідження є процеси створення інфраструктури аплікації з базами даних.

Предметом дослідження є методи автоматизації розгортання інфраструктури аплікації та подальша робота з базами даних.

Актуальність теми полягає в тому, що по суті робота з даними, а саме їх зберігання та подальша обробка лежить в основі як економічних моделей світових гігантів електронної комерції так і переважної більшості представників малого та середнього бізнесу. Крім показників фінансової ефективності, компанії в сфері роботи з даними повинні неухильно дотримуються імплементації положень “Загального регламенту про захист даних” (GDPR – General Data Protection Regulation), що робить роботу з базами даних, в яких ця інформація зберігається, надзвичайно важливим критичним компонентом успішного бізнесу.

У результаті виконання роботи було розглянуто та проаналізовано процеси створення інфраструктури аплікації з базами даних, в якості прикладу реалізації концепції моделі управління було створена аплікація для демонстрації роботи системи.

Отримані під час роботи результати можливо покращити впровадженням нових програмних компонентів які відносяться до керування змінами в програмному забезпеченню конфігурацій та оптимізації існуючого функціоналу.

ABSTRACT

Thesis: 136 p., 6 tabl., 39 fig., 2 appendices, 21 sources.

CLOUD ENVIRONMENT, RELATIVE AND NON-RELATIVE DATABASES,
RESOURCE MANAGEMENT MODEL, DATA STORAGE

The purpose of this work is to increase the efficiency and reliability of the resource management model of heterogeneous databases in a cloud environment based on applications with different types of databases and web interface.

The object of research is the processes of creating an application infrastructure with databases.

The subject of research is the methods of automation of application infrastructure deployment and further work with databases.

The relevance of the topic is that, in fact, working with data, namely their storage and further processing is the basis of both economic models of global e-commerce giants and the vast majority of small and medium-sized businesses. In addition to financial performance indicators, data companies must strictly adhere to the provisions of the General Data Protection Regulation (GDPR), which contains provisions and requirements for the processing of personal information of data subjects that work with databases. The data in which this information is stored is an extremely important critical component of a successful business.

As a result of the work, the processes of creating an application infrastructure with databases were considered and analyzed, as an example of the implementation of the concept of the management model, an application was created to demonstrate the system.

The results obtained during the work can be improved through the introduction of new software components related to change management in software configuration and optimization of existing functionality.

ЗМІСТ

Перелік скорочень.....	8
Вступ.....	9
1 Порівняльний аналіз на основі дослідження предметної області.....	12
1.1 Актуальність проблеми.....	12
1.2 Загальна характеристика моделей управління ресурсами БД.....	16
1.3 Програмні комплекси для управління і підтримки ресурсів БД в хмарних середовищах.....	24
1.4 Проблеми впровадження системи управління ресурсами БД.....	30
1.5 Порівняльний огляд та аналіз існуючих на даний момент програмних комплексів для управління і підтримки ресурсів БД в різних хмарних середовищах.....	34
1.6 Висновки.....	42
2 Моделі управління ресурсами гетерогенних баз даних.....	43
2.1 Загальна характеристика моделей управління.....	43
2.2 Опис обраної моделі та її особливості.....	49
2.3 Висновки.....	55
3 Розробка програмної реалізації моделі управління ресурсами БД.....	56
3.1 Вибір засобів для розробки програмної реалізації.....	56
3.2 Проектування і реалізація структури баз даних.....	57
3.3 Розробка функціоналу програмної реалізації.....	63
3.4 Робота з інтерфейсом програмної реалізації.....	70
3.5 Висновки.....	73

	7
4 Функціонально-вартісний аналіз програмного продукту.....	74
4.1 Постановка завдання.....	74
4.2 Обґрунтування функцій та параметрів дослідження.....	76
4.3 Визначення коефіцієнтів значимості параметрів.....	81
4.4 Аналіз рівня якості варіантів реалізації функцій програмного продукту.....	85
4.5 Економічний аналіз варіантів розробки програмного продукту.....	86
4.6 Вибір кращого варіанту програмного продукту техніко-економічного рівня.....	91
4.7 Висновки.....	92
Висновки.....	93
Перелік джерел посилання.....	94
Додаток А лістинг програми.....	96
Додаток Б ілюстративний матеріал доповіді.....	128

ПЕРЕЛІК СКОРОЧЕНЬ

БД — база даних

СУБД — система управління базою даних

GDPR — General Data Protection Regulation — загальний регламент про захист даних

HIPPA — Health Insurance Portability and Accountability Act — Закон про переносимість та підзвітність медичного страхування

ANSI — American National Standards Institute — Американський національний інститут стандартів

SQL — Standart Query Language — структурована мова запитів

API — Application Programming Interface — прикладний програмний інтерфейс

IaaS — Infrastructure as a Service — інфраструктура як послуга

PaaS — Platform as a Service — платформа як послуга

SaaS — Software as a Service — програмне забезпечення як послуга

ACID — акронім слів atomicity-атомарність, consistency-узгодженість, isolation-ізолюваність, durability-довговічність

IaC — Infrastructure as Code — інфраструктура як код

YAML — рекурсивний акронім YAML Ain't Markup Language — YAML не мова розмітки

SSH — Secure SHell — безпечна оболонка

HCL — HashiCorp Configuration Language — синтаксис конфігурацій Terraform

ВСТУП

Значну роль в сучасних бізнесових системах багатьох компаній світового рівня займають правила та особливості роботи з даними у хмарних середовищах. Існує просте та чітке визначення що таке хмарне середовище, створене Національним інститутом стандартів і технологій США. По суті, хмарні обчислення - це спосіб використання інфраструктури, що має п'ять однаково важливих складових. Перше, користувач отримує обчислювальні ресурси на вимогу. Хмарні провайдери надають доволі простий інтерфейсу для отримання необхідної обчислювальної потужності, сховище та мережу без втручання людини. Друге, користувач отримує доступ до цих ресурсів через мережу з будь - якого місця. Третє, постачальник має великий об'єм ресурсів які він розподіляє їх серед клієнтів. Користувачам більше не потрібно знати і піклуватися про точне фізичне розташування цих ресурсів. Четверте, ресурси еластичні. Якщо користувачу потрібно більше ресурсів, він їх може отримати надзвичайно швидко і так же само швидко обмежити масштаб їх використання. П'яте, споживачі платять лише за те, що вони використовують або бронюють для використання. Якщо вони припиняють використовувати ресурси, вони припиняють платити. Ця сукупність ознак є визначенням сучасного хмарного середовища [18].

Використання в хмарному середовищі технологій баз даних зумовлено потребою бізнесу в ефективних засобах зберігання і обробки інформації. Під інформацією розуміють будь-які відомості про будь-яку подію, сутність, процес, які є об'єктом певних операцій: передачі, перетворення, зберігання або використання. У сучасному світі інформація та її використання стає фактором, що визначає ефективність будь-якої сфери діяльності. Будь-які адміністративні дії

вимагають попередньої оцінки поточної ситуації та можливих перспектив її зміни. І якщо раніше ситуація оцінювалася декількома факторами, то зараз цих факторів, можливо, сотні або тисячі, при цьому ситуація змінюється не протягом року, а за кілька хвилин. Природно, що обґрунтованість прийнятих рішень повинна зрости багаторазово, тому що реакція на неправильні рішення буде більш критичною, ніж раніше. Інформаційні задачі, які виникають в таких ситуаціях мають певні особливості як-то збереження даних складної структури з простими алгоритмами обробки які працюють з великими обсягами оброблюваної інформації. Інформаційна система - взаємопов'язана сукупність засобів, методів і персоналу, використовуваних для зберігання, обробки та видачі інформації в інтересах досягнення поставленої мети. У більшості випадків сьогодні під інформаційною системою розуміють автоматизовані інформаційні системи, які дозволяють автоматизувати діяльність, підвищити якість і достовірність оброблюваної інформації. Інформаційна система виконує функції збору та збереження даних, а також обробку та маніпулювання даними. Вона забезпечує структурування даних і управління ними і має життєвий цикл який складається з послідовних етапів як планування, збір і аналіз вимог, проектування, реалізація, тестування, супроводження. Ефективність роботи інформаційної системи залежить від таких складових як проект та реалізації бази даних, проект та реалізація застосувань, супроводження інформаційної системи. Інформаційну основу самих інформаційних систем складають дані що зберігаються в ній. У більшості випадків це база даних, а для управління даними використовують система управління базами даних [9].

База даних - структурована сукупність даних, організована за певними правилами. База даних є фундаментальним компонентом інформаційної системи та використовуються для зберігання інформації про об'єкти будь-якої предметної області, самі ж дані можна визначити як інформацію, зафіксовану у певній формі,

яка придатна для подальшої обробки, зберігання і передачі. Склад бази даних містить не тільки дані, що зберігаються, але і опис самої бази даних. Опис бази даних належить вже до метайнформації.

Використання сучасних систем управління базами даних дозволяє отримати такі переваги як скорочення надмірності в збережених даних разом з усуненням можливості виникнення суперечливості збережених даних, також забезпечується дотримання стандартів в поданні даних, прийнятих в даній предметній області з обов'язковим виконанням умов безпеки даних та підтримки їх цілісності. Можливо визначити процедури перевірки, що виконуються при операціях запам'ятовування, при цьому має забезпечуватися незалежність даних і програм, тобто можливість розширення додатків незалежно від бази даних і навпаки можливість розширення бази даних без впливу на додатки [21].

Об'єктом даної роботи є дослідження та розробка програмного комплексу для управління ресурсами баз даних в хмарному середовищі. Перший розділ даної роботи присвячений аналізу предметної області. Другий розділ описує загальні характеристики моделей управління та обрану модель. Третій розділ описує реалізацію функціоналу, четвертий розділ містить функціонально-вартісний аналіз програмного продукту.

1 ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ МЕХАНІЗМІВ НА ОСНОВІ ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Активне використання компаніями інфраструктури хмарних провайдерів для зберігання і обробки важливої інформації окрім суто економічної доцільності має також на меті дотримання існуючих законодавчих норм щодо роботи з даними, основні з яких це Загальний регламент про захист даних GDPR(General Data Protection Regulation) та Закон про переносимість та підзвітність медичного страхування HIPPA(Health Insurance Portability and Accountability Act).

Загальний регламент про захист даних - це новий регламент, який замінив попередню Директиву про захист даних з метою гармонізації захисту, наданого всім громадянам ЄС, щодо способу використання їх особистих даних як всередині ЄС, так і за його межами. Регламент був прийнятий, підписаний та опублікований в Офіційному віснику Європейського Союзу 27 квітня 2016 року, і він набув чинності з травня 2018 року. GDPR визначає персональні дані як дані, що стосуються живих осіб, яких можна ідентифікувати за цими даними. Спеціальні категорії персональних даних (також відомі як чутливі персональні дані) включають дані, що стосуються расового або етнічного походження, політичних поглядів, релігійних або філософських переконань, членство в профспілках, біометричні дані, генетичні дані, статева активність або сексуальна орієнтація, інформація щодо здоров'я(медична інформація), адміністративні або кримінальні провадження та санкції.

GDPR встановлює такі основні принципи використання персональних даних:

Законність, чесність та прозорість - персональні дані обробляються законно, чесно та прозоро. Обмеження мети - персональні дані збираються для конкретних, явних та законних цілей і не обробляються далі способом, несумісним із цими цілями. Мінімізація даних - персональні дані мають бути адекватними, актуальними та обмеженими тим, що необхідно для цілей, для яких вони обробляються. Точність - персональні дані повинні бути точними і, при необхідності, оновлюватися; особисті дані, які є неточними щодо цілей, для яких вони обробляються, стираються або виправляються негайно. Обмеження зберігання(утримання) - персональні дані зберігаються у формі, яка дозволяє ідентифікувати осіб не довше, ніж це необхідно для цілей, для яких обробляються персональні дані. Цілісність і конфіденційність - персональні дані обробляються таким чином, який забезпечує належну безпеку персональних даних, включаючи захист від несанкціонованої чи незаконної обробки та від випадкової втрати, знищення чи пошкодження. Обробка персональних даних означає будь - яку операцію або набір операцій, які виконуються над персональними даними або над набором персональних даних, будь то автоматизованими або іншими засобами, такими як збір, запис, організація, структурування, зберігання, адаптація чи зміна, пошук, консультації, використання, розкриття шляхом передачі, розповсюдження або іншим наданням доступу, вирівнювання або поєднання, обмеження, стирання або знищення. GDPR надає особам такі права як: право на прозорість персональних даних, право на виправлення, право на обмеження обробки, право на доступ, право на заперечення, право не підлягати автоматизованому прийняттю рішень, включаючи профілювання, право на стирання ("право бути забутим"), право на перенесення даних.

GDPR докорінно змінив спосіб роботи з персональними даними, бо порушення підприємствами деяких положень GDPR може призвести до штрафів у

розмірі до 20 млн євро або 4% світового річного обороту за попередній фінансовий рік в залежності від ступеня і серйозності порушення.

Закон про переносимість та підзвітність медичного страхування HIPAA в США був введений у 1996 році для встановлення стандартів захисту конфіденційності медичної інформації. Регламент регулює використання та передачу "Захищеної інформації про здоров'я" (Personal Health Information), яка є індивідуально ідентифікованою інформацією про здоров'я, що зберігається у будь-якому форматі. Крім того, 26 березня 2013 року набрало чинності додаток HIPAA для посилення конфіденційності та безпеки інформації про здоров'я людини. Як відповідати вимогам HIPAA: обмежити використання та передачу медичної інформації про особу (осіб); дотримуватися гарантій політики та процедур організації щодо конфіденційності та безпеки особистої інформації; повідомляти про будь-які підозрілі випадки, що стосуються медичної інформації.

Актуальність питання дотримання вищенаведених законодавчих норм даних не підлягає обговоренню оскільки серйозне порушення та штраф можуть призвести до повного банкрутства компанії-порушника, отже логічним та вигідним рішенням, принаймні щодо фізичного доступу до даних та процедур їх обробки буде проведення їх в захищеному середовищі хмарних провайдерів, в яких неухильне виконання цих норм буде організоване на найвищому можливому технологічному рівні.

За даними компанії Gartner станом на липень 2021 року лідери хмарних технологій представлені на рисунку 1.1.



Source: Gartner (July 2021)

рисунок 1.1 - Дослідження компанії Gartner щодо хмарних провайдерів

Зважаючи на важливість питання роботи з даними та доступність продуктів для дослідження в рамках дипломної роботи були вибрані лідери ринку, а саме: Amazon Web Service, Microsoft Azure Platform, Google Cloud Platform.

1.2 Загальна характеристика моделей управління ресурсами БД

Важливою умовою організації роботи з базою даних є забезпечення незалежності додатків від даних, тому що при зміні системи, а також для забезпечення обслуговування користувачів необхідно виконувати роботу зі зміни способів зберігання даних у базі даних, способів доступу до даних, зміни структур даних та форматів та стосунки між ними. Якщо не використовувати спеціальних підходів під час написання додатків для введення програми опису методів доступу, зберігання, форматів даних, то зміна бази даних для цих випадків потребуватиме коригування тексту програми, що вимагає значних витрат.

Трирівнева модель архітектури системи управління базою даних, запропонована Американським національним інститутом стандартів (ANSI - American National Standards Institute) показана на рисунку 1.2.

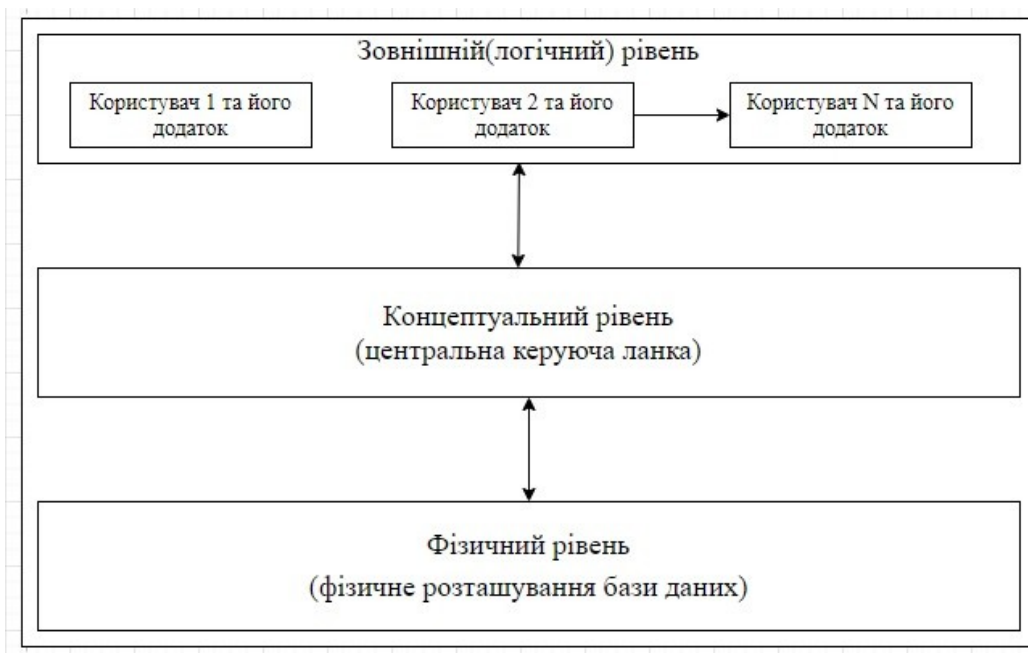


рисунок 1.2 - Трирівнева модель архітектури СУБД

Основна перевага цієї архітектури є забезпечення логічної і фізичної незалежності при роботі користувача з наявними даними.

Логічна незалежність передбачає можливість зміни однієї програми без коригування інших додатків, що працюють з цією базою даних.

Фізична незалежність передбачає можливість зміни способу зберігання, розташування або перенесення даних на носіях (на фізичному рівні) без впливу на працездатність всіх додатків які вже працюють з базою даних.

Іншими словами, якщо відбувається будь-яка зміна в розташуванні даних на носіях, це ніяк не відбивається на роботі програм зовнішнього рівня. І навпаки, якщо відбувається будь-яка доповнення додатків зовнішнього рівня, то змін на фізичному рівні не відбувається. Перехід від одного рівня абстрагування до наступного і складає в загальному вигляді процес проектування бази даних.

Важливою складовою в концепції бази даних є такі узагальнені категорії як дані і модель даних.

Дані - це набір конкретних значень та параметрів, що характеризують об'єкт.

Модель даних - це деяка абстракція, що прикладається до конкретних даних, яка дозволяє користувачам трактувати їх як інформацію, в сенсі розуміння інформації, що містить не тільки дані, але і з описанням взаємозв'язку між ними. В моделі розрізняють три головні складові, це структурна частина, керуюча частина та класи обмежень цілісності даних.

Моделювання даних - це процес створення логічного представлення структури бази даних.

Система управління базами даних - це пакет програмного забезпечення для створення та управління базами даних. Існує багато різних типів систем баз даних на основі того, як вони керують структурою бази даних. Бази даних зазвичай мають одну з двох основних форм: однофайлова або плоска файлова база даних,

та багатофайлова реляційна або структурована база даних. Плоска база даних зберігає дані у звичайному текстовому файлі, при цьому кожен рядок тексту зазвичай містить один запис. Розділювачі, такі як коми або табуляції, відокремлюють поля. База даних з плоскими файлами використовує просту структуру і не може містити декілька таблиць і відносин [19].

Реляційна база даних містить кілька таблиць даних із рядками та стовпцями, які пов'язані між собою через спеціальні поля ключів. Ці бази даних є більш гнучкими, ніж плоскі файлові структури, і забезпечують функціональність для читання, створення, оновлення та видалення даних. Реляційні бази даних використовують структуровану мову запитів (SQL) - стандартну програму користувача, яка забезпечує простий інтерфейс програмування для взаємодії з базами даних.

Типи відносин у реляційній базі даних

- 1) У реляційних баз даних існує чотири типи відносин:
- 2) один до одного - де один запис таблиці відноситься до іншого запису в іншій таблиці
- 3) один до багатьох - де один запис таблиці відноситься до кількох записів в іншій таблиці
- 4) багато до одного - де більш ніж один запис таблиці відноситься до іншого запису таблиці
- 5) багато до багатьох - де кілька записів стосуються більш ніж одного запису в іншій таблиці

Ці відносини формують функціональні залежності всередині бази даних. Деякі поширені приклади реляційних баз даних включають MySQL, Microsoft SQL Server, Oracle тощо.

Системи управління базами даних можна поділити на чотири типи:

- 1) реляційні базами даних

- 2) ієрархічні бази даних
- 3) мережеві бази даних
- 4) об'єктно-орієнтовані бази даних

Реляційна модель є одним з найбільш широко використовуваних механізмів організації баз даних. Вона нормалізує дані та впорядковує їх як логічно незалежні таблиці. Спільні дані зображують відносини між різними таблицями. Оскільки дані в таблиці можуть посилатися на подібні дані в іншій таблиці, це зберігає надійність зв'язків між ними. Це називається реферальною цілісністю, що є критичною концепцією в цій моделі бази даних. Серед інших переваг: легке розширення, оскільки нові дані можна додавати без зміни існуючих записів-масштабованість та безпека даних, яка є надзвичайно важливою, коли обмін даними ґрунтується на конфіденційності. Наприклад, можливо поділитися певними привілеями щодо даних або швидко заблокувати конфіденційну інформацію, представлена на рисунку 1.3.

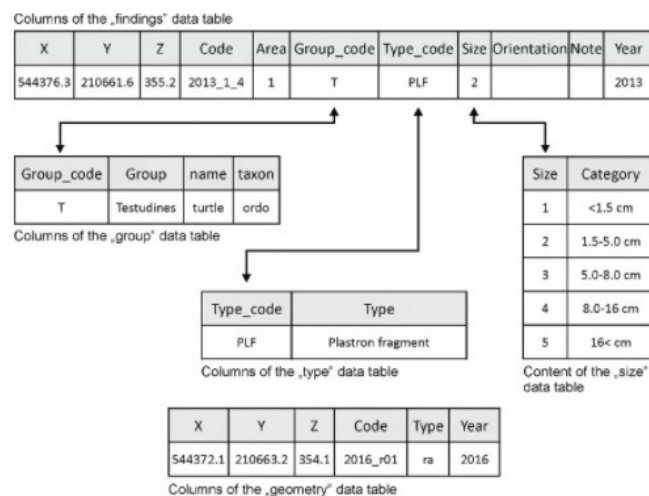


Рисунок 1.3 — Реляційна модель

Ієрархічна модель бази даних нагадує деревну структуру, подібну до архітектури папок у комп'ютерній системі. Взаємовідносини між записами визначені заздалегідь, один до одного, між «батьківськими та дочірніми» вузлами. Вони вимагають від користувача проходження ієрархії для доступу до необхідних даних. Через обмеження такі бази даних можуть обмежуватися у використанні. Ідея ієрархічних моделей баз даних корисна для певного типу зберігання даних, вони даних були популярні на початку проектування баз даних. Хоча деякі моделі IBM та Microsoft все ще використовуються, багато інших типів баз даних використовують більш гнучкі моделі для адаптації до більш складних типів управління даними. Ієрархічні моделі мають сенс, коли основний фокус збору інформації зосереджений на конкретній ієрархії, представлена на рисунку 1.4.

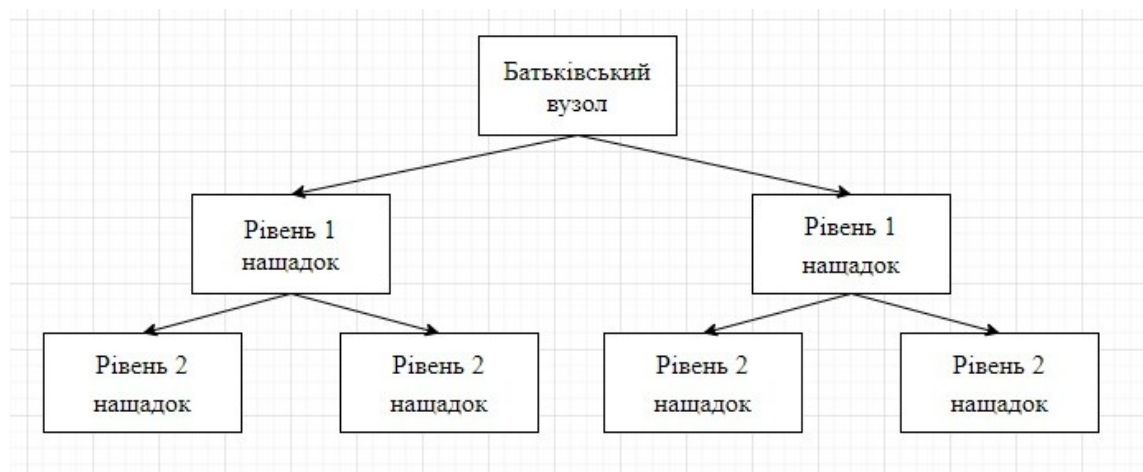


Рисунок 1.4— Ієрархічна модель

Моделі мережеских баз даних також мають ієрархічну структуру. Однак, замість використання ієрархії дерева з одним батьком, ця модель підтримує багато-багато відносин, оскільки дочірні таблиці можуть мати кілька батьків. Модель мережевої бази даних дозволяє кожному запису мати кілька батьківських і декількох дочірніх записів, які при візуалізаціях утворюють веб-подібну структуру

мережевих записів. Ця властивість наявності декількох посилань застосовується двома способами: схему та саму базу даних можна розглядати як узагальнений графік типів записів, які пов'язані між собою. Основною перевагою мережевої бази даних є те, що вона дозволяє більш природне моделювання відносин між записами або сутностями, на відміну від ієрархічної моделі, БД представлена на рисунку 1.5.

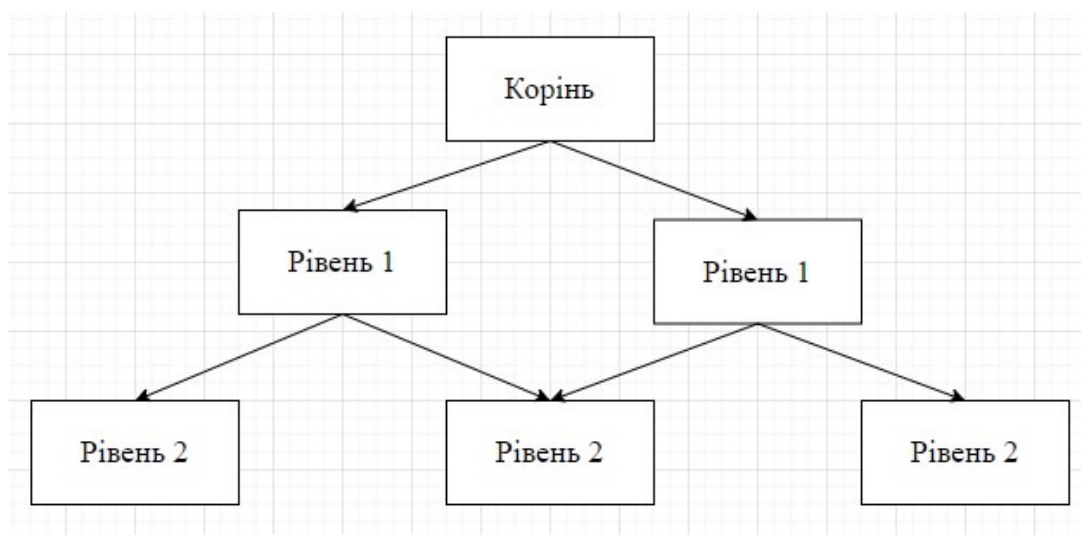


Рисунок 1.5— Мережева модель

В моделі об'єктно-орієнтованих баз даних інформація представлена у вигляді об'єктів, при цьому можливі різні типи відносин між двома або більше об'єктами [20]. Такі бази даних використовують для розробки об'єктно-орієнтовану мову програмування. Основною особливістю об'єктно-орієнтованих баз даних є можливість визначення об'єктів, які відрізняються від звичайних об'єктів баз даних. Об'єкти в об'єктно-орієнтованій базі даних посилаються на можливість розробки продукту, а потім визначають і називають його, на об'єкт

можна посилатися або викликати його пізніше як одиницю. Це дуже схоже на об'єкти, що використовуються в об'єктно-орієнтованому програмуванні. Простий приклад об'єктно-орієнтованого програмування - це автомобільний двигун. Він складається з кількох частин: головного блоку циліндрів, вихлопної системи, впускного колектора тощо. Кожен з них є окремим компонентом але при механічній обробці та кріпленні в один об'єкт окремі запчастини починають називаються двигуном. Аналогічно, при програмуванні можна визначити кілька компонентів, таких як вертикальна лінія, що перетинає перпендикулярну горизонтальну лінію, хоча при цьому обидві лінії мають градування. Потім цей об'єкт можна позначити як графік, модель БД представлено на рисунку 1.6.

Object-Oriented Model

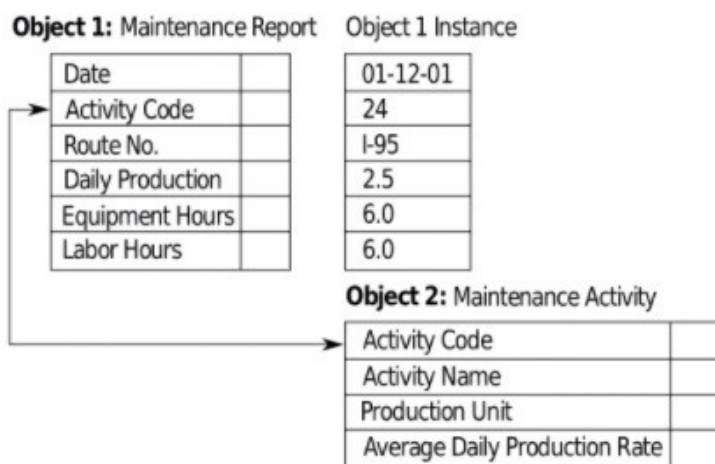


Рисунок 1.6— Об'єктно-орієнтована модель

NoSQL або нереляційні бази даних. Популярна альтернатива реляційним базам даних, бази даних NoSQL мають різноманітні форми і дозволяють зберігати та маніпулювати великими обсягами неструктурованих та слабоструктурованих

даних. Приклади включають сховища ключ-значення та сховища документів. Більшість нереляційних баз даних включені до таких веб-сайтів, як Google, Amazon та Facebook. Ці веб - сайти щодня представляють безліч нових аплікацій з мільйонами користувачів, тому вони не зможуть впоратися з великими стрибками трафіку за допомогою існуючих рішень системи управління базами даних. Оскільки реляційна база даних не може впоратися з проблемою, вони перейшли на новий вид системи управління базами даних, яка здатна обробляти дані веб масштабу нереляційним способом. Цікавим аспектом нереляційної бази даних, такої як NoSQL є масштабованість. NoSQL використовує систему BASE (basically available, soft-state, eventually consistent). Нереляційні бази даних відмовилися від табличної форми рядків і стовпців, які використовують реляційні бази даних на користь спеціалізованих фреймворків для зберігання даних, до яких можна отримати доступ за допомогою спеціальних API запитів. Важливим елементом цих баз даних є стійкість, модель представлена на рисунку 1.7

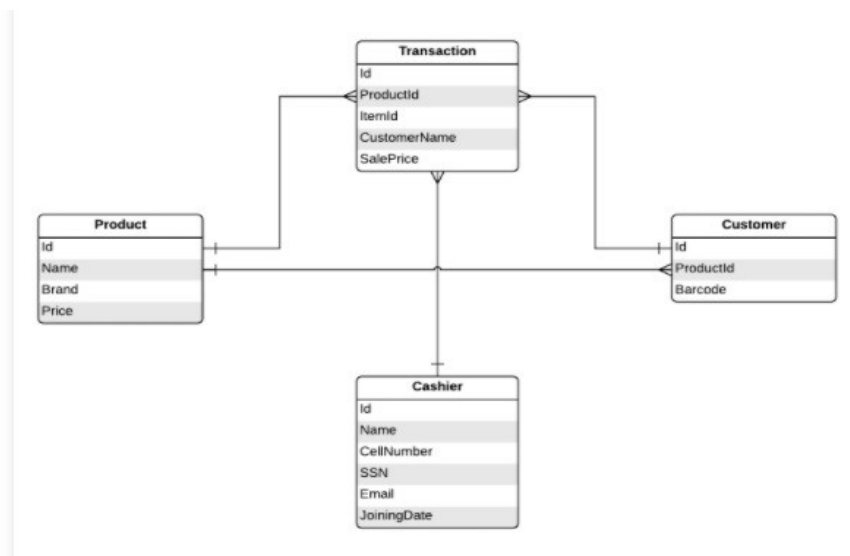


Рисунок 1.7—NoSQL модель

1.3 Програмні комплекси для управління і підтримки ресурсів БД в хмарних середовищах

Управління ресурсами хмарного середовища - це спосіб, яким адміністратори контролюють та організовують усі продукти та послуги, які працюють у хмарі: користувачів та контроль доступу, дані, програми та послуги. Мова йде про надання адміністраторам можливості отримувати доступ до необхідних ресурсів, автоматизувати потрібні для роботи процеси і вносити необхідні корективи, а також відстежувати використання та вартість, зберігаючи при цьому гнучкість і масштабованість у випадку швидкоплинної зміни ситуації. Інструменти керування хмарою допомагають адміністраторам контролювати всі типи хмар, як-то загальнодоступну хмару(public cloud), приватну хмару(private cloud), гібридну(hybrid cloud), багатохмарну(multicloud) структуру та іншу діяльність в хмарі починаючи від розгортання ресурсів, відстеження їх використання до інтеграції даних, управління за допомогою Кубернетіса(Kubernetes) та відновлення після збоїв. Вони забезпечують адміністративний контроль над інфраструктурою, платформами, програмами, ресурсами провайдера хмарних послуг, а також даними, які разом створюють хмару [17].

Програмне забезпечення для управління хмарою зазвичай розгортається у наявних хмарних середовищах така як віртуальна машина, що містить базу даних та сервер. Сервер спілкується з інтерфейсами програмування прикладних програм (API) для з'єднання бази даних та віртуальних, хмарних чи контейнерних ресурсів, які підтримують хмару та взаємодіють із нею. База даних збирає інформацію про ефективність віртуальної чи хмарної інфраструктури та надсилає аналізи до веб - інтерфейсу, де адміністратори хмари можуть візуалізовувати та аналізувати роботу ресурсів. Адміністратори також можуть передавати команди назад у хмару, які

виконуються віртуальним сервером. Цей взаємозв'язок залежить від роботи операційної системи яка не тільки керує всіма різними технологіями, з яких і складаються хмари, але й розгортає потрібні нам інструменти управління. Операційна система повинна працювати з будь-якою хмарою, якою б ми не користувалися, в переважній більшості випадків це буде дистрибутив Лінукс(Linux). Варто зауважити що управління хмарами(cloud management) - це не хмарні обчислення(cloud computing). Провайдери хмарних послуг надають правила, а управління хмарами(cloud management) їх виконує. Хмари по суті це просто пули віртуальних ресурсів, сховища, додатків та служб, якими керує програмне забезпечення, тому до них можна отримати доступ за запитом. Хмарне управління - це поєднання програмного забезпечення, автоматизації, політики, управління та людей, які визначають, яким чином та для кого ці послуги хмарних обчислень стають доступними [10].

Сервіси та компоненти, які повинні надавати програмні комплекси для управління і підтримки ресурсів баз даних в хмарних середовищах.

1) Інтеграція з вже існуючими рішеннями. Платформа керування хмарами (cloud management platform) має бути налаштована відповідно до потреб користувача: вона повинна відповідати вимогам операційної системи, додатків, надання ресурсів, безпеки, зберігання, стандартних операційних середовищ та будь - чого іншого, що працює у хмарі. Вони також повинні бути гнучкими та здатними швидко адаптуватися до змін.

2) Автоматизація задач. Автоматизація дозволяє звільнити продуктивні робочі години від рутинних повсякденних завдань і робочих процесів, що відразу може бути виражено у збільшенні ефективності та зменшенні кількості необов'язкових помилок. Автоматизація є життєво важливою для будь - якої платформи управління хмарою.

3) Візуалізація витрат. Користуватися хмарними службами легко та просто, тому можливо дуже швидко перевищити свій фінансовий ліміт. Платформа хмарного управління повинна забезпечувати точне управління витратами, моніторинг, прогнозування та складання звітності.

4) Підтримка гібридного хмарного та багатохмарного середовищ. За необхідності компанія повинна мати можливість роботи з усіма типами хмар: приватними, публічними та гібридними. Кожна платформа керування хмарами повинна мати можливість безперебійно та безпечно підтримувати всі типи хмар у різних середовищах.

5) Широкий спектр управління. Можливості управління та спектр доступних налаштувань є життєво важливими для будь-якої платформи управління хмарами, щоб залишатися гнучкою та адаптованою перед частими змінами та оновленнями. Потрібно зберігати контроль над усім, тоді як політики доступу безпосередньо керують доступом користувачів до ресурсів, конфігурацій та можливостей платформи.

Хмарні провайдери пропонують широкий спектр послуг, адаптованих до різних вимог клієнтів, функціональний опис який має закінчення aaS(as a Service)

Три основних наборів сервісів це IaaS, PaaS, SaaS які позначають відповідно інфраструктуру як послугу (Infrastructure as a Service), платформу як послугу (Platform as a Service) і програмне забезпечення як послугу (Software as a Service).

IaaS пропонує вам максимальну гнучкість у розміщенні спеціальних програм, а також загальний центр обробки даних для зберігання даних.

PaaS найчастіше будується на платформі IaaS, щоб зменшити потребу в системному адмініструванні. Це дозволяє зосередитися на розробці додатків замість управління інфраструктурою.

SaaS пропонує готові до використання, готові рішення, які відповідають певним бізнес-потребам (таким як веб-сайт або електронна пошта). Більшість сучасних платформ SaaS побудовані на платформах IaaS або PaaS.

Можливо почати з однієї моделі сервісу хмарних обчислень або задіяти усі три: це залежить від розміру та складності бізнесу, приклад використання сервісів в контексті вищезазначених понять продемонстровано на рисунку 1.8.

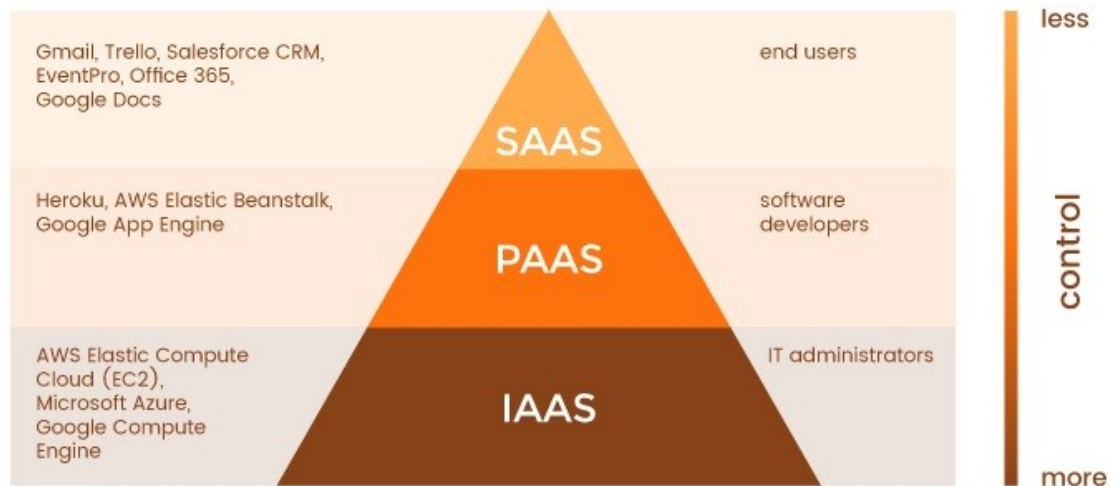


Рисунок 1.8— Приклади хмарних сервісів

При роботі з базами даних ми в основному користуємось ресурсами які належать до парадигм IaaS (інфраструктура як послуга) та PaaS(платформа як послуга). IaaS це обчислювальна модель, яка пропонує підприємствам та приватним особам ресурси за запитом через хмару. Власну інфраструктуру складно масштабувати, особливо в ситуації коли попит на послуги зростає дуже різко. Для компаній-початківців, якщо їх діяльність не обмежена специфічними вимогами національного законодавства, IaaS є гарним вибором. Ресурси IaaS

пропонуються як окремі послуги, тому користувачі можуть вибрати виключно те, що їм потрібно. Оскільки хмарний провайдер керує інфраструктурою, користувачі можуть зосередитися на встановленні, налаштуванні та управлінні програмним забезпеченням та забезпеченні безпеки своїх даних.

По суті, IaaS - це можливість за вимогою отримати майже нескінченно масштабовані обчислювальні ресурси як сервіс через Інтернет, при чому користувачі вони платять лише за те, що вони використовують або бронюють до використання.

Основні переваги сервісу IaaS

1) Економічна доцільність. Оскільки ресурси IaaS використовуються на вимогу, а користувачі мають сплачувати лише за обчислювання, сховища та мережеві ресурси які фактично використовуються, витрати на IaaS досить передбачувані і їх достатньо легко визначити та утримувати.

2) Ефективність. IaaS ресурси доступні користувачам, саме тоді, коли вони їх потребують. В результаті користувачі не відчують затримки з розширенням інфраструктури та не витрачають ресурси на потужності які потрібні вкрай рідко або тимчасово.

3) Надійність. IaaS не має єдиної точки відмови. Навіть якщо будь - який компонент апаратних ресурсів вийде з ладу, сервіс залишиться доступним.

4) Масштабованість. Однією з найбільших переваг IaaS у хмарних обчисленнях є можливість швидко масштабувати ресурси відповідно до потреб користувача.

PaaS (платформа як послуга) - це категорія послуг хмарних обчислень, що дозволяє клієнтам надавати, створювати, запускати та керувати модулем, що включає обчислювальну платформу та одну або кілька програм без етапу побудови та підтримки інфраструктури. PaaS в основному використовується

розробниками, які створюють програмне забезпечення або програми, оскільки розробникам або користувачам готових рішень надається платформа для створення програмного забезпечення. Це означає, що розробникам не потрібно починати з нуля при створенні програм, що економить їм багато часу і коштів на написання великого коду.

Переваги PaaS

- 1) Масштабованість
- 2) Побудований на технології віртуалізації
- 3) Легкість управління
- 4) Розробнику можливо зосередитися на творчій стороні розробки додатків, без етапів управління оновленнями програмного забезпечення або безпеки.

1.4 Проблеми впровадження системи управління ресурсами БД

Існує декілька загальновизнаних проблем з управлінням ресурсами баз даних які змінюються по мірі того як змінюються потреби самого бізнесу. Зростає концентрація на безпеці та питанні щодо зберігання та аналітики даних яка стає все складніше. Велика кількість доступних опцій робить складним навіть сам процес вибору ефективного рішення для управління базами даних, оскільки зі збільшенням кількості варіантів зростають і витрати часу та зусиль на збір інформації.

Дана тематика потребує окремого аналізу, оскільки за останні роки обсяги даних зросли, а способи обробки та використання даних постійно змінюються.

Основні проблеми управління базами даних:

1) Зростаюча складність технологій зберігання та обробки інформації.

У міру розвитку ринку баз даних багатьом користувачам важко оцінити та вибрати правильне рішення. Існують реляційні бази даних, об'єктно-орієнтовані бази даних та бази даних NoSQL, не кажучи вже про велику кількість постачальників, які пропонують свої власні можливості.

2) Обмеження масштабованості

Справа в тому, що все програмне забезпечення має обмеження щодо масштабованості та використання ресурсів, включаючи сервери баз даних. Далекоглядні компанії, стурбовані можливостями обробки транзакцій, знають, що компоненти каталогів, архітектура бази даних і навіть операційні системи та конфігурація обладнання впливають на масштабованість, тому необхідно враховувати цілий ряд аспектів ще на етапі архітектурної розробки концепту бази даних.

3) Збільшення обсягів даних

Оскільки кількість отриманих та зібраних даних зростає експоненціально, компанії зазнають серйозних труднощів намагаючись не відставати від збільшення можливостей по їх обробці. Дослідження показують, що за останні два роки ми створили більше даних ніж за всю попередню історію людства, а розвиток систем та алгоритмів для їх обробки поки що не відповідає вимогам ринку або існуючі рішення є занадто дорогими.

4) Безпека даних

Бази даних є ключовими компонентами ІТ-систем багатьох компаній, що зберігають критичні публічні та приватні дані. Сучасна база даних має забезпечувати найвищий рівень безпеки який легко налаштовувати, мати при цьому можливість швидко повідомити користувачів про інциденти і надійно захищати дані як у стані зберігання та при передачі. Порушення цілісності даних може коштувати штрафів у розмірі до 20 млн євро або 4% світового річного обороту, не кажучи вже про втрату ділової репутації та довіри клієнтів.

5) Децентралізоване управління даними

Хоча децентралізоване управління даними має переваги, це також викликає і додаткові труднощі в питанні розподілу даних, вибору найкращого методу та ступеня децентралізації. Дані потрібно обробляти таким чином, щоб нічого не було втрачено. Навіть у нереляційній базі даних або в розподіленому кластері повинні існувати гарантії ACID(atomicity, consistency, isolation, durability) як у всій базі даних.

6) Продуктивність

Бази даних завжди стикалися з жорсткими критеріями ефективності. Кількість даних з якими нам доводиться працювати буде збільшуватися і задовольнити своїх користувачів ми повинні обробляти їх швидше. Якісні показники продуктивності передбачають розгортання з хорошими дефолтними налаштуваннями. Для забезпечення високої продуктивності необхідно створювати

базу даних яка не залежить від обладнання, маючи можливість роботи на різного класу машинах.

Незважаючи на певні існуючі складнощі користувачі мають працювати з існуючими рішеннями, тому існують декілька перевірених практикою рекомендацій для правильного вибору:

1) Встановлення критерію прийняття рішень.

Першим кроком на шляху до вибору правильного рішення є створення об'єктивного стандарту, за допомогою якого можна оцінити наявні варіанти. Звичайно, кожен користувач матиме різні критерії та методи їх оцінки, як-то вартість володіння, простота використання, функціональність, легкість адміністрування бази даних, масштабованість тощо.

2) Відповідність рішення бізнес-задачам.

Вибір конкретної технології бази даних повинен враховувати бізнес-цілі, як-то кількість даних які ми розраховуємо збирати, наскільки швидко це потрібно робити, яким чином отримувати доступ та аналізувати їх. Кожен бізнес відрізняється, тому тут немає єдиної відповіді. Більше критеріїв на цьому етапі допоможе найбільш точно вибрати необхідні технології.

3) Взаємодія з існуючими рішеннями.

По можливості, згідно принципу “бритви Оккама”, потрібно уникати використання нових технологій просто заради їх використання без зміни функціоналу існуючого рішення, оскільки це може призвести до неочікуваних наслідків.

4) Навантаження на апаратні ресурси.

Фактично будь-який вибір буде оцінюватися за продуктивністю бази даних або за тим, як швидко вона надає інформацію користувачам. Важливо пам'ятати, що навантаження може різко коливатися в залежності від дня, години чи навіть хвилини. Мета повинна полягати в тому, щоб максимально можливе навантаження

оброблялося без оновлення ресурсів.

Дуже важливо вибрати правильне рішення з використанням саме усіх вищенаведених параметрів, оскільки система управління ресурсами бази даних стане одним із основних програмних ресурсів від яких буде залежати бізнес.

1.5 Порівняльний огляд та аналіз існуючих на даний момент програмних комплексів для управління і підтримки ресурсів БД в різних хмарних середовищах

В даній роботі будуть розглянуті рішення від наступних хмарних провайдерів:

- 1) Google Cloud Platform
- 2) Amazon Web Service
- 3) Microsoft Azure Platform

Google Cloud Platform

Google Cloud Platform має багато варіантів зберігання даних, які задовольняють практично всі випадки обробки та зберігання даних, рисунок 1.9. Основні варіанти зберігання даних та бази даних: Google Cloud Storage, Google Cloud SQL, Google Cloud Spanner, Cloud Datastore та Google Cloud Bigtable [1].

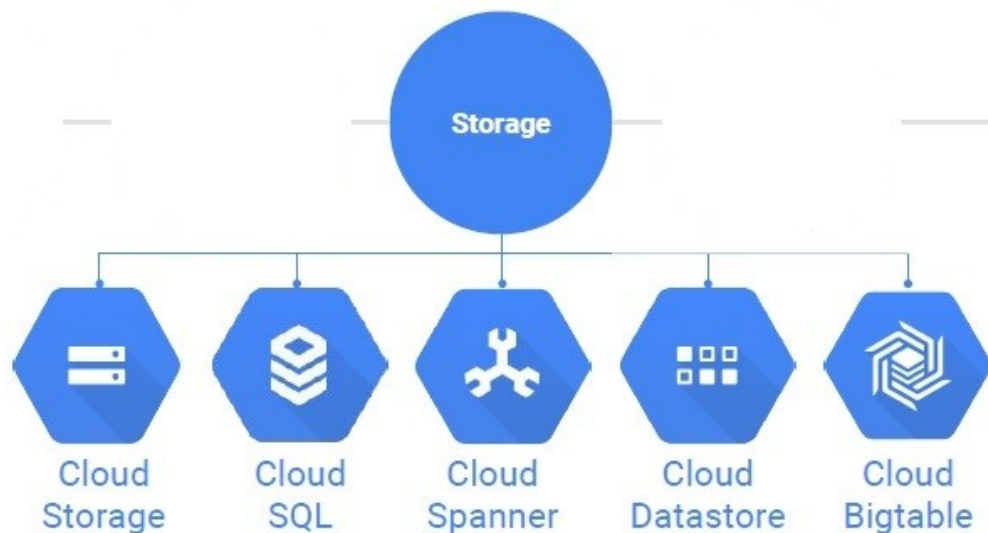


Рисунок 1.9 — Моделі роботи з даними Google Cloud Platform

1) Cloud Storage. Фактично це BLOB(Binary Large Object) сховище. Бінарний набір даних що розглядається як єдиний, неперервний об'єкт. У контексті баз даних BLOB це спеціальний тип даних, призначений, передусім, для зберігання зображень, аудіо і відео. Повноцінний RESTful сервіс для зберігання та доступу до даних в інфраструктурі Google. Послуга поєднує продуктивність та масштабованість хмари Google із розширеними можливостями безпеки та спільного доступу. Висока продуктивність та просте адміністрування, дані шифруються при зберіганні та при передачі.

2) Cloud SQL – це веб-сервіс парадигми PaaS який дозволяє створювати, налаштовувати та використовувати реляційні бази даних, які знаходяться у хмарі Google. Це повністю керований сервіс, який підтримує, керує та адмініструє бази даних, дозволяючи користувачам зосередитися на своїх додатках та послугах. Бази даних MySQL та PostgreSQL як сервіс, автоматична реплікація, бекап, вертикальне масштабування (читання та запис), горизонтальне масштабування (читання).

3) Cloud Spanner – це повністю керований, критично важливий сервіс реляційної бази даних. Він розроблений для забезпечення масштабованої бази даних онлайн -обробки транзакцій OLTP(Online Transaction Processing) з високою доступністю та цілісністю даних у глобальному масштабі. Спосіб організації бази даних, при якому система працює з невеликими за розмірами транзакціями що йдуть великим потоком і при цьому клієнту потрібний від системи максимально швидкий час відповіді.

4) Cloud Datastore – це повністю кероване нереляційне сховище даних. Воно надає багатий набір щодо можливостей запитів, підтримує ACID(atomicity, consistency, isolation, durability) та автоматично масштабується вгору та вниз у відповідь на завантаження. Сервіс може масштабуватися для підтримки програми з 1000 або 10 мільйонами користувачів.

5) Cloud Bigtable – це швидкий, повністю керований, високомасштабований сервіс баз даних NoSQL. Він призначений для збору та збереження даних від 1 TB до сотень PB. Bigtable ідеально підходить для зберігання дуже великих обсягів даних у сховищі "ключ-значення", підтримуючи високу пропускну здатність читання та запису з низькою затримкою для швидкого доступу до великих обсягів даних. Пропускна здатність масштабується лінійно, можливо збільшити QPS(queries per second) запити в секунду, додавши ноди Bigtable. Bigtable побудований на основі перевірених рішень інфраструктурою, яка підтримує роботу таких продуктів Google як Google Search та Google Maps.

Порівняльна таблиця можливостей продуктів Google представлена на рисунку 1.10.

	Cloud Datastore	Bigtable	Cloud Storage	Cloud SQL	Cloud Spanner
Type	NoSQL document	NoSQL wide column	Blobstore	Relational SQL for OLTP	Relational SQL for OLTP
Transactions	Yes	Single-row	No	Yes	Yes
Complex queries	No	No	No	Yes	Yes
Capacity	Terabytes+	Petabytes+	Petabytes+	Up to ~10 TB	Petabytes
Unit size	1 MB/entity	~10 MB/cell ~100 MB/row	5 TB/object	Determined by DB engine	10,240 MiB/row

рисунок 1.10 — Порівняльна таблиця характеристик продуктів Google

Amazon Web Service

Amazon Web Service на даний момент є беззаперечним лідером світового ринку серед хмарних провайдерів, пропонуючи декілька перевірених часом рішень для роботи з даними, таких як Amazon S3, Amazon RDS, Amazon DynamoDB, Amazon Redshift, Amazon ElastiCache [2].

1) Amazon S3 – це сервіс зберігання об’єктів, який пропонує провідну в галузі масштабованість, доступність даних, безпеку та продуктивність. Це означає, що клієнти будь-якого розміру та галузі можуть використовувати його для зберігання та захисту будь-якої кількості даних для різноманітних випадків використання, таких як веб-сайти, мобільні програми, резервне копіювання та відновлення, архівування, корпоративні програми, пристрої Інтернету речей та аналітика даних тощо. Amazon S3 надає прості у використанні функції управління з гнучкими можливостями щодо налаштування засобів керування доступом відповідно до конкретних вимог користувача.

2) Amazon RDS – це веб-сервіс, яка полегшує налаштування, роботу та масштабування реляційної бази даних у хмарі. Він забезпечує економічно ефективну, з можливістю зміни розміру реляційну базу даних та керує загальними завданнями адміністрування баз даних. RDS – це не база даних, по суті це сервіс який управляє базами даних, таких як:

3) Amazon Aurora реляційна база даних, створений компанією Amazon, який поєднує швидкість та надійність комерційних баз високого класу з простотою та економічністю відкритих баз даних. Amazon стверджує, що Aurora в 5 разів швидше, ніж RDS MySQL.

4) MySQL система управління базами даних з відкритим вихідним кодом, яка використовує SQL для доступу до даних, що зберігаються в її системі.

5) PostgreSQL – це ще одна система управління базами даних з відкритим кодом, яка використовує SQL для доступу до даних.

6) Microsoft SQL Server – це система управління реляційними базами даних, яка була розроблена Microsoft у 2005 році для корпоративного середовища.

7) Oracle BYOL - це об'єктно-реляційна система управління базами даних, розроблена компанією Oracle Inc.

8) MariaDB - це розвинений спільнотою форк MySQL.

9) Amazon DynamoDB – це безсерверна база даних для програм, яким потрібна висока продуктивність. NoSQL база даних ключ-значення яка забезпечує надзвичайну продуктивність у будь-якому масштабі. Це повністю керована, мультирегіональна стійка база даних із вбудованими принципами безпеки, резервним копіюванням та відновленням. DynamoDB може обробляти більше 10 трильйонів запитів на день і може підтримувати піки більше 20 мільйонів запитів на секунду.

10) Amazon Redshift – швидке, просте та економічно ефективне рішення для зберігання даних. Redshift дозволяє робити запити та комбінувати екзабайти структурованих та напівструктурованих даних у сховищі даних, операційній базі даних використовуючи стандартний SQL. Redshift дозволяє легко зберігати результати ваших запитів назад до сховища даних S3, використовуючи відкриті формати, такі як Apache Parquet, дозволяючи робити додаткову аналітику в поєднанні з іншими аналітичними сервісами, таких як Amazon EMR, Amazon Athena та Amazon SageMaker.

11) Amazon ElastiCache – керована служба зберігання даних у пам'яті (in-memory data store services), яка полегшують розгортання, управління та масштабування популярних сховищ даних з відкритим кодом.

Microsoft Azure Platform

Microsoft Azure Platform надає окремі рішення щодо зберігання даних та роботи з базами даних [3].

Зберігання даних - Azure Storage пропонує декілька типів масштабованих та високодоступних сховищ. В багатьох випадках визначити найкраще рішення для вимог компанії або звичайного користувача не просто, оскільки існує два типи облікових записів сховища, п'ять типів сховищ, та ще три рівні зберігання даних.

Типи сховищ: Blob Storage, Table Storage, File Storage, Queue Storage, Disk Storage, рисунок 1.11.

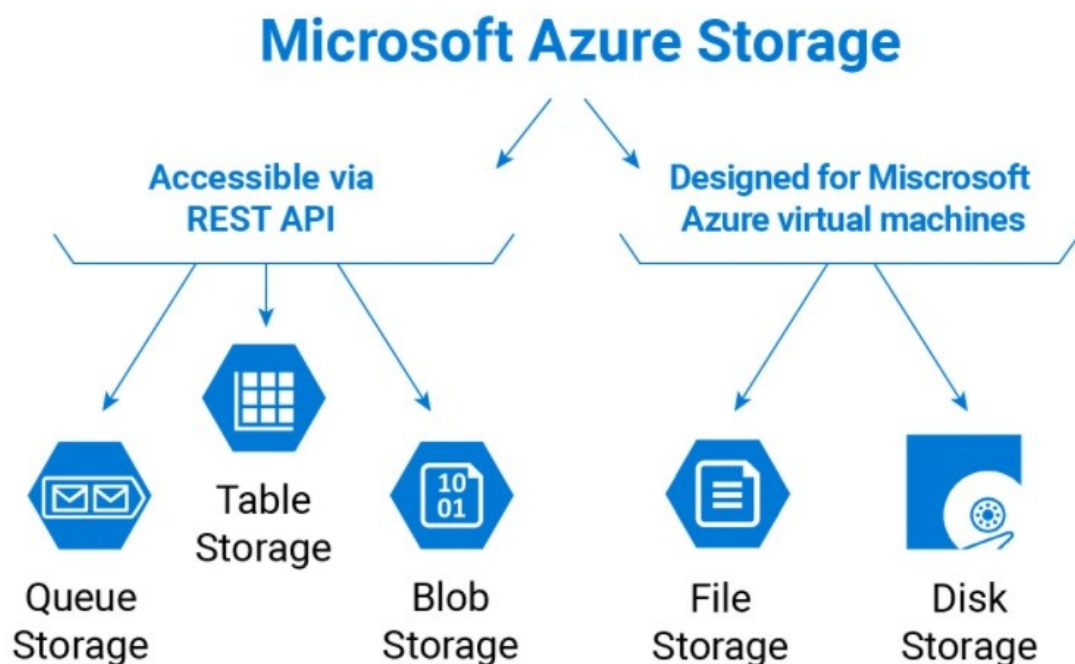


рисунок 1.11 — Моделі роботи з даними Microsoft Azure Storage

Blob Storage - рішення для зберігання величезної кількості неструктурованих даних, таких як текстові або двійкові дані. Дані, що зберігаються не повинні відповідати будь-яким конкретним критеріям. Далі він поділяється на 3 типи:

1) Block blob - це найдешевший та найпростіший спосіб зберігання даних у сховищі BLOB. Ми зберігаємо файл як єдиний неділимий об'єкт.

2) Page blob - це базова інфраструктура віртуальної машини Microsoft Azure, в основному використовується для зберігання зображень на дисках віртуальної машини користувача.

3) Append blobs використовується для зберігання логів(logs), які потрібно часто оновлювати.

Table Storage використовується для зберігання структур та даних NoSQL. Table Storage може бути корисним для розробників, оскільки дозволяє зберігати терабайти даних із порівняно невеликими витратами на зберігання, в якості недоліку має дорогий доступ до файлів.

Queue Storage використовуються в тих випадках, коли сервіси мають спілкуватися між собою. Цей тип зберігання діє як посередник або буфер між двома службами. Queue Storage надає механізм стійкості до помилок, коли відбувається втрата даних через тривалий час процесу.

File Storage використовується для підтримки віртуальних машин Microsoft Azure Virtual Machines. File Storage дозволяє зберігати і отримувати доступ до файлів з різних віртуальних машин.

Disk Storage є аналогією на сховище яке розташоване на персональному комп'ютері. Він дозволяє створювати диски для віртуальної машини користувача та надавати доступ до нього лише з цієї віртуальної машини, надаючи можливість використання двох типів дисків та моделі керування для зберігання дисків – це HDD, які є дешевими, але повільнішими та SSD, які швидші але дорожче.

Бази даних - Azure пропонує вибір повністю керованих реляційних та NoSQL баз даних що охоплює як власні рішення так і рішення з відкритим кодом, що відповідає потребам сучасних розробників програм, характеристики баз даних наведені на рисунку 1.12.

- 1) Azure SQL Database
- 2) Azure SQL Managed Instance

- 3) SQL Server on Virtual Machines
- 4) Azure Database for PostgreSQL
- 5) Azure Database for MySQL
- 6) Azure Database for MariaDB
- 7) Azure Cosmos DB
- 8) Azure Cache for Redis
- 9) Azure Managed Instance for Apache Cassandra

	Azure SQL Database	Azure SQL Managed Instance	SQL Server on Virtual Machines	Azure Database for PostgreSQL	Azure Database for MySQL	Azure Database for MariaDB	Azure Cosmos DB	Azure Cache for Redis
Relational Database	✓	✓	✓	✓	✓	✓		
Non-Relational Database (NoSQL)							✓	
In-Memory Database								✓
Data Models	Relational	Relational	Relational	Relational	Relational	Relational	Multi-Model: Document Wide-column Key-Value Graph	Key-Value
Hybrid	✓	✓	✓	✓ (Hyperscale)				
Serverless Compute	✓						✓	
Storage Scale Out	✓ (Hyperscale)			✓ (Hyperscale)			✓	✓
Compute Scale Out	✓ (Hyperscale - read-only)			✓ (Hyperscale)			✓	✓
Distributed Multi-Master Writes (Write data to different regions)							✓	✓ (Coming Soon)
OSS Based Service (Community edition and open extension support)				✓	✓	✓		✓

рисунок 1.12 — Порівняльна таблиця характеристик хмарних рішень для зберігання та обробки даних Microsoft

1.6 Висновки

У даному розділі були розглянуті питання щодо загальних характеристик баз даних та особливості їх функціоналу, детально проаналізовані моделі управління ресурсами баз даних. Розкриті також питання програмних комплексів для управління і підтримки ресурсів баз даних та проблеми впровадження системи управління ресурсами баз даних. Висвітлена проблематика впровадження системи управління базами даних та наведені рішення щодо їх усунення. Проведений детальний порівняльний огляд та аналіз існуючих на даний момент програмних комплексів для управління і підтримки ресурсів БД в різних хмарних середовищах.

2 МОДЕЛІ УПРАВЛІННЯ РЕСУРСАМИ ГЕТЕРОГЕННИХ БАЗ ДАНИХ

2.1 Загальна характеристика моделей управління

Вибір інфраструктурних рішень, який надають хмарні провайдери наразі дуже великий і більш того, існуюча тенденція така що надання інфраструктури або імплементація готових пакетних рішень невпинно рухається в сторону автоматизації цих процесів. Приблизно десять років тому починає виникати концепція “інфраструктура як код” – IaC(infrastructure as code) яка радикально змінює спосіб створення та управління сучасною IT-інфраструктурою. Сьогодні створення більш гнучкої, економічно ефективної та безпечної інфраструктури можна здійснити за допомогою програмного коду в рази швидше та надійніше ніж за допомогою графічного інтерфейсу користувача [13].

Можливість зберігати цей код у системі контролю версій SCM(source code management) з наданням певної системи різного рівня доступу до цього коду при інтенсивній командній роботі спростило розробку програмного забезпечення і тестування рішень, рисунок 2.1.

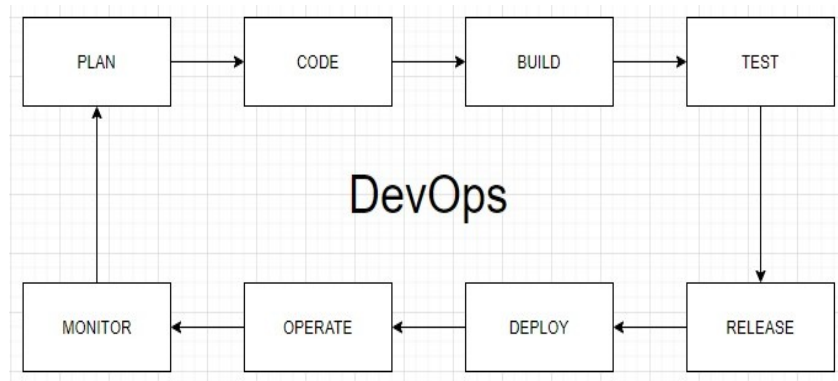


рисунок 2.1 — Методологія DevOps

Розглядаючи інфраструктуру як програмне забезпечення та дані, команди розробників також можуть застосувати цілий ряд методів DevOps(software — Моделі роботи з даним та методології Agile, які сприяють швидкому та якісному робочому процесу створення продукту [11].

Команди можуть використовувати найкращі практики, такі як безперервна інтеграція (continuous integration CI), безперервна доставка (continuous delivery CD) та проводити розробку на основі тестів (test-driven development TDD), рисунок 2.2.

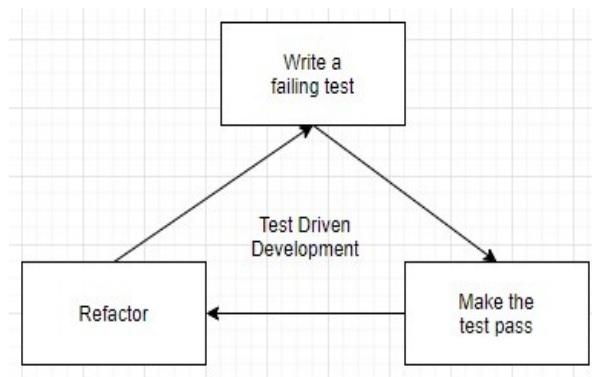


рисунок 2.2 — Система розробки на основі тестів

Організації також можуть оптимізувати використання оркестровки розгортання, бібліотек автоматизованого тестування для покращення відмовостійкості своїх систем та існуючих рішень. Завдяки швидкій еволюції, IaC стає більш доступним сьогодні завдяки широкому спектру інструментів, доступних для автоматизації всього процесу надання ресурсів [14].

Програмні рішення які надають хмарні провайдери в парадигмі IaC.

AWS CloudFormation

Сервіс призначений для користувачів, які працюють у нативній хмарі. Станом на серпень 2021 року CloudFormation підтримує 775 типів ресурсів AWS у 160 сервісах AWS. CloudFormation дозволяє користувачам моделювати свою інфраструктуру у файлі шаблону JSON або YAML. Сервіс також додає функції автоматизації, які допомагають із розгортанням ресурсів та дозволяє платити лише за використані ресурси, а не за саму послугу. З шаблоном, налаштованим відповідно до специфікацій потрібного додатка, CloudFormation буде обробляти решту завдань самостійно, дозволяючи легко встановити безпечну модель інфраструктури на будь – якому рівні складності з багатьох шаблонів, доступних у CloudFormation.

Azure Resource Manager

Інструмент IaC від Microsoft доступний для користувачів у вигляді Azure Resource Manager. За допомогою цього засобу користувачі можуть створювати інфраструктуру та обробляти залежності в одному безперервному циклі за допомогою шаблонів Azure Resource Manager(ARM). Ресурси, для яких використовується шаблон декларативно описані в JSON і можливо оголосити декілька ресурсів Azure в одному шаблоні ARM. Оскільки шаблони ARM також ідемпотентні, є можливість повторно використовувати той самий шаблон потрібну кількість разів і завжди отримувати однакові результати. Дашборд VSTS дозволяє візуально відстежувати всі білди та релізи, отримуючи швидкий огляд загального стану робочого середовища та якості шаблонів.

Google Cloud Deployment Manager

Інструмент автоматизації від Google, який базує своє виконання на файлах конфігурації YAML та шаблонах JINJA2 або PYTHON у межах хмарної платформи Google. Він також дозволяє визначати ресурси та синхронно їх задеплоїти. Є можливість повністю описати скриптом розгортання з

можливостями автоматичного масштабування та балансування навантаженням. Google CDM також підтримує попередній перегляд.

Рішення в парадигмі IaC які надають можливість роботи з різними хмарними провайдерами.

Terraform є універсальним інструментом парадигми IaC. Подібно до Google CDM, Terraform також підтримує попередній перегляд змін. Він дозволяє визначати ресурси та інфраструктуру у зручному для розуміння коді (human-readable) через декларативні файли конфігурації у форматі YAML, керуючи життєвим циклом інфраструктури. Переваги Terraform це керування інфраструктурою на багатьох хмарних платформах, проста мова конфігурації яка допомагає швидко писати інфраструктурний код, можливість відстежувати зміни ресурсів безпосередньо під час розгортання [4].

Ansible вважається найпростішим способом автоматизації надання, налаштування та управління IT-інфраструктурою. Ansible дозволяє користувачам виконувати плейбуки для створення та управління необхідними ресурсами інфраструктури, підключись до серверів та виконуючи команди через SSH. Код написаний у YAML, можливо написати власні модулі та плагіни Ansible для задоволення конкретних потреб [5]. Цей інструмент має графічний інтерфейс для налаштування та моніторингу, схема роботи представлена на рисунку 2.8.

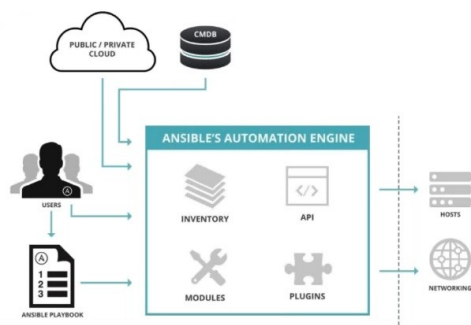


рисунок 2.8 — Схема роботи Ansible

Chef є доволі популярним інструментом IaC який використовує процедурну мову, в якій користувач повинен написати код і покроково вказати спосіб досягнення бажаного стану. Від користувача залежить визначення оптимального процесу розгортання. Використовуючи свій Ruby-based DSL(domain specific language) Chef дозволяє створювати “рецепти”(recipes) та “кулінарні книги”(cookbooks) які вже визначають ті самі точні кроки, необхідні для досягнення бажаної конфігурації програм, маючи в своїй основі шаблони та колекції шаблонів, готових до використання. Одна кулінарна книга виконує одне завдання, але вона може запропонувати низку різних конфігурацій сервера на основі залучених ресурсів (наприклад, веб-програма з базою даних матиме два рецепти, по одному для кожної частини що зберігаються разом). Завдяки підтримці API щодо роботи з хмарними сервісами, Chef також дуже добре працює з іншими інструментами IaC, включаючи Terraform, а також з багатьма хмарними середовищами. Цей інструмент управління інфраструктурою призначений для розгортання та моделювання масштабованого та безпечного процесу автоматизації інфраструктури у будь-якому середовищі [7].

Puppet – інструмент управління конфігурацією з відкритим кодом, він також використовує Ruby-based DSL, де користувач може описати бажаний кінцевий стан вашої інфраструктури. Основна відмінність Puppet від Chef полягає в тому, що він використовує декларативний підхід, де користувач вирішує як виглядатиме потрібна конфігурація, а потім Puppet її впроваджує. Puppet відноситься до інструментів IaC, які використовуються для швидкої та безпечної розгортки інфраструктури, пропонуючи модулі, які допомагають розширити функціональність програмного забезпечення. Puppet інтегрується майже з усіма провідними постачальниками хмарних платформ та керує центрами обробки даних для кількох значних компаній, таких як Reddit, Dell та Google, працюючи у всіх ОС, маючи один з найдосконаліших інтерфейсів серед IaC інструментів [6].

Він також відстежує в інфраструктурі зміни, які відхиляються від визначеного кінцевого стану, і автоматично їх виправляє. Це інструмент, розроблений спеціально для системних адміністраторів завдяки значній підтримці ентерпрайз та комюніті.

Saltstack – це інструмент управління конфігурацією з відкритим вихідним кодом на базі Python, який пропонує просте рішення для забезпечення, розгортання та налаштування інфраструктури на будь-якій платформі з дуже високою швидкістю. SaltStack зосереджується на автоматизації інфраструктури, безпеки та мережі всередині організації. Це безпечне та економічно ефективне рішення IaC, яке забезпечує автоматизацію та оркестровку, даючи можливість автоматично виявляти проблеми з тригерами подій та повертатися у потрібний стан при необхідності. У інструменту навіть є підтримка SSH, яка може підтримувати режим без агентів. Saltstack має планувальник, який дозволяє вказати, як часто сервери повинні запускати код [8].

2.2 Опис обраної моделі та її особливості

Проаналізувавши вищенаведену інформацію щодо можливостей та особливостей інструментів IaC для реалізації інфраструктури був обраний Terraform.

Інструменти IaC дозволяють керувати інфраструктурою за допомогою файлів конфігурації, а не за допомогою графічного інтерфейсу користувача. IaC дозволяє створювати, змінювати та управляти своєю інфраструктурою безпечним, послідовним та повторюваним способом, визначаючи конфігурації ресурсів які можна модифікувати, використати повторно та надавати спільний доступ. Terraform є продуктом компанії HashiCorp як інструмент створення та підтримки інфраструктури, він дозволяє визначати ресурси та інфраструктуру у зручних для розуміння декларативних файлах конфігурації та керувати життєвим циклом інфраструктури, принцип роботи представлений на рисунку 2.12.

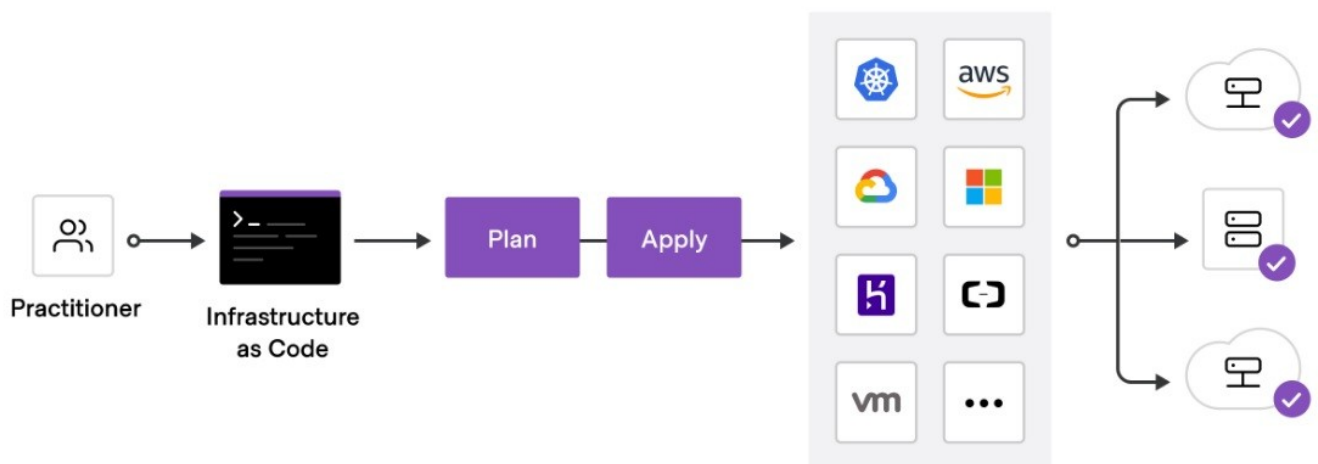


рисунок 2.12 — Принцип роботи Terraform

Використання Terraform має ряд переваг перед ручним управлінням інфраструктурою:

- 1) Terraform може керувати інфраструктурою на кількох хмарних платформах.
- 2) Зручна та зрозуміла мова конфігурації дозволяє швидко писати інфраструктурний код.
- 3) Можливо відстежувати зміни ресурсів під час розгортання.
- 4) Працює з VCS

Terraform дозволяє керувати будь-якою інфраструктурою через плагіни які називаються провайдерами, дозволяючи йому взаємодіяти з хмарними платформами та іншими сервісами через прикладні програмні інтерфейси (application programming interface - API). HashiCorp та спільнота Terraform написали станом на серпень 2021 року понад 1000 плагінів провайдерів для управління ресурсами серед яких Amazon Web Services, Azure, Google Cloud Platform та багато інших. Усіх постачальників всіх платформ та послуг можливо знайти у реєстрі Terraform Registry. Можливо додати власного провайдера, який буде визначати окремі одиниці інфраструктури, наприклад інстанси або приватні мережі як ресурси. Користувач може об'єднувати ресурси різних постачальників у багаторазові конфігурації Terraform, які називаються модулями і керувати ними. Синтаксис конфігурацій Terraform називається HashiCorp Configuration Language (HCL). Мова конфігурації Terraform є декларативною, тобто вона описує бажаний кінцевий стан інфраструктури, на відміну від процедурних мов програмування, які вимагають покрокових інструкцій для виконання завдань. Terraform також автоматично обчислює залежності між ресурсами для створення або знищення їх у правильному порядку.

Інструменти IaC можна розділити на 5 категорій:

- 1) скрипти
- 2) засоби управління конфігурацією
- 3) засоби шаблонізації серверів
- 4) засоби оркестрації
- 5) засоби ініціалізації ресурсів

Модель роботи Terraform - це інструмент з відкритим вихідним кодом від компанії HashiCorp, написаний на мові програмування Go. Код на Go компілюється в єдиний двійковий файл, який дозволяє розгорнути інфраструктуру прямо з будь-якої машини або чи сервера (взагалі будь-якого іншого комп'ютера), і для всього цього не потрібно ніякої додаткової інфраструктури. Все завдяки тому, що виконуваний файл terraform робить від імені користувача API-виклики до одного або кількох провайдерів, таким як AWS, Azure, Google Cloud, DigitalOcean, OpenStack і так далі. Це означає, що Terraform використовує інфраструктуру, яку ці провайдери надають для своїх API-серверів, а також їх механізми аутентифікації, які ми вже застосовуємо (наприклад, API-ключі для AWS). Щоб Terraform розумів які саме API-виклики потрібно робити ми створюємо текстові файли з конфігурацією, в яких описуємо потрібну нам інфраструктуру. У концепції «інфраструктура як код» ці файли грають роль коду. Ось приклад конфігурації Terraform:

```
resource "aws_instance" "example" {  
  
    ami          = "ami-0c55b159cbfaffe1f0"  
  
    instance_type = "t2-micro"  
  
}  
  
resource "google_dns_record_set" "a" {
```

```

name          = "demo.google-example.com."

managed_zone = "example-zone"

type          = "A"

rrdatas       = [aws_instance.example.public_ip]

ttl           = 300
}

```

Цей фрагмент дає команду Terraform виконати API-виклики до двох провайдерів: до AWS для розгортання серверу і до Google Cloud, щоб створити DNS-запис, який вказує на IP-адресу сервера з AWS. Terraform дозволяє використовувати єдиний простий синтаксис для розгортання взаємопов'язаних ресурсів в декількох різних хмарах. Всю інфраструктуру - сервери, бази даних, балансувальник навантаження, топологію мережі тощо, можливо описати в конфігураційних файлах Terraform і зберегти їх в системі управління версіями. Потім цю інфраструктуру можна буде розгорнути за допомогою певних команд, таких як `terraform apply`. Terraform проаналізує код, перетворить його в послідовність API-викликів до потрібних хмарних провайдерів і виконає ці API-виклики від імені користувача максимально ефективним чином. Коротко підсумовуючи вищеописане, Terraform - це утиліта, яка перетворює вміст конфігураційних файлів в API-виклики до хмарних провайдерів.

Такий підхід робить командну роботу швидкою та ефективною, оскільки в ситуації коли розробник хоче змінити інфраструктуру, то замість того щоб робити це вручну на серверах він редагує конфігураційні файли Terraform, перевіряє їх за допомогою автоматичних тестів і розбору коду, фіксує оновлений код в системі

управління версіями і потім виконує команду `terraform apply`, щоб зробити необхідні для розгортання змін API-виклики, рисунок 2.13.

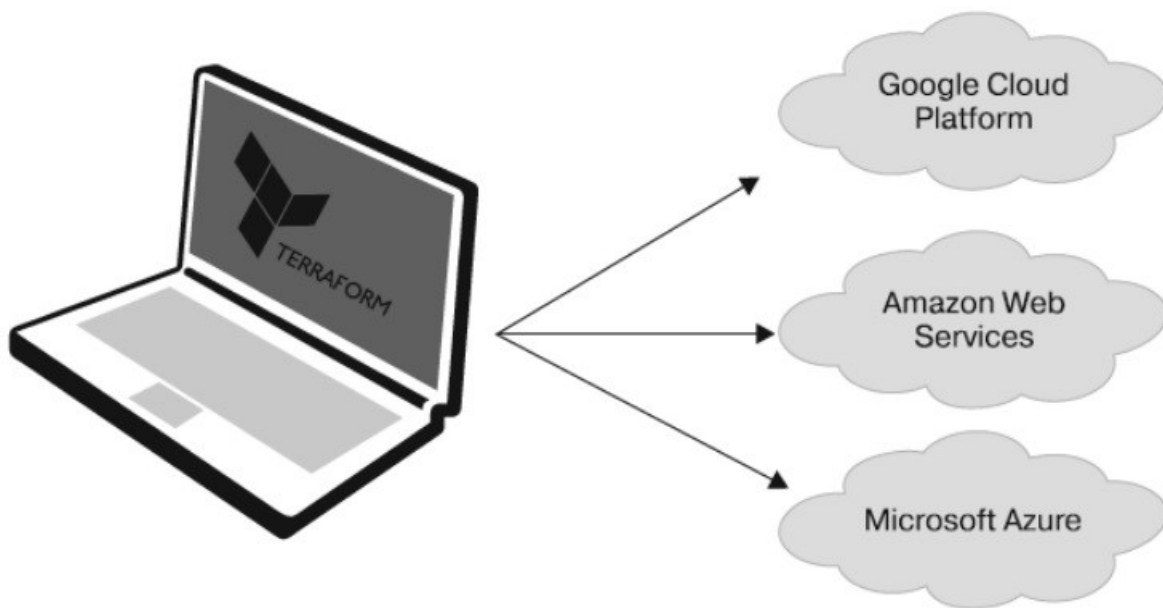


рисунок 2.13 — Схема роботи Terraform з різними хмарними провайдерами

Окремої уваги потребує питання щодо переносимості між хмарними провайдерами. Terraform підтримує безліч різних хмарних провайдерів і було б логічно припустити, що якщо ми використовували Terraform для опису кількох серверів, баз даних, балансувальника навантаження та іншої інфраструктури в AWS, то можливо і розгорнути це все в іншому хмарі, такому як Azure або Google Cloud. На практиці це питання виявляється не настільки простим. Розгорнути ідентичну інфраструктуру в різних хмарах з одним зразком коду неможливо, оскільки різняться сама інфраструктура, яка надається хмарними провайдерами. Сервери, балансувальник навантаження і бази даних, пропоновані в AWS, Azure і

Google Cloud, сильно розрізняються з точки зору можливостей, конфігурації, управління, безпеки, масштабованості, доступності, спостереження і так далі. На даний час не існує простого і прозорого способу подолати ці відмінності, особливо з огляду на те, що деякі функції одного хмарного провайдера часто відсутні у всіх інших.

Підхід, який використовується в Terraform, дозволяє писати код для кожного провайдера окремо користуючись його унікальними можливостями, при цьому всередині для всіх провайдерів застосовують ту саму мову, інструментарій та методики IaC.

Завдяки декларативним підходу Terraform код завжди описує поточний стан інфраструктури, даючи можливість миттєво визначити, що зараз розгорнуто і як воно налаштоване, не піклуючись про історію змін або синхронізації. Це значно полегшує написання коду, придатного для повторного використання, оскільки тепер не потрібно враховувати поточний стан навколишнього світу та зосередитися лише на описі потрібного нам стану, завдяки чому кодова база Terraform зазвичай залишається компактною і зрозумілою. Звичайно, у декларативного підходу є свої недоліки. Відсутність доступу до повноцінного мови програмування позначається на виразності. Наприклад, деякі види зміни інфраструктури, такі як розгортання без простою, доволі складно висловити в чисто декларативному стилі. Аналогічно обмежені можливості опису логіки коду (такі як умовні вирази і цикли) роблять непростим написання універсального коду, який можна застосовувати повторно. Terraform надає ряд потужних примітивів: вхідні і вихідні змінні, модулі, `create_before_destroy`, `count`, тернарний синтаксис і вбудовані функції. Все це дозволяє писати чистий модульний код навіть на декларативні мови.

2.3 Висновки

У даному розділі була розглянута загальна характеристика моделей управління ресурсами гетерогенних баз даних від хмарних провайдерів AWS, Google, Microsoft, відповідно це ресурси AWS CloudFormation, Google Cloud Deployment Manager, Azure Resource Manager. Також в розділі розглянута концепція “інфраструктура як код” та наведені наступні інструменти з оглядом принципів роботи такі як: Terraform, Ansible, Chef, Puppet, Saltstack та проведений їх аналіз. Також було розглянуто вибрану модель реалізації та розглянуті її особливості.

3 РОЗРОБКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МОДЕЛІ УПРАВЛІННЯ РЕСУРСАМИ БД

3.1 Вибір засобів для розробки програмної реалізації

Для реалізації програмного комплексу було вибрано 2 категорії інструментів ІаС, як-то Terraform для створення інфраструктури та скрипт, написаний на BASH. Програмний код написаний за допомогою IDE Visual Studio Code, який має підтримку плагінів мови YAML з підсвічуванням синтаксису, що допомагає уникнути синтаксичних помилок, рисунок 3.1.

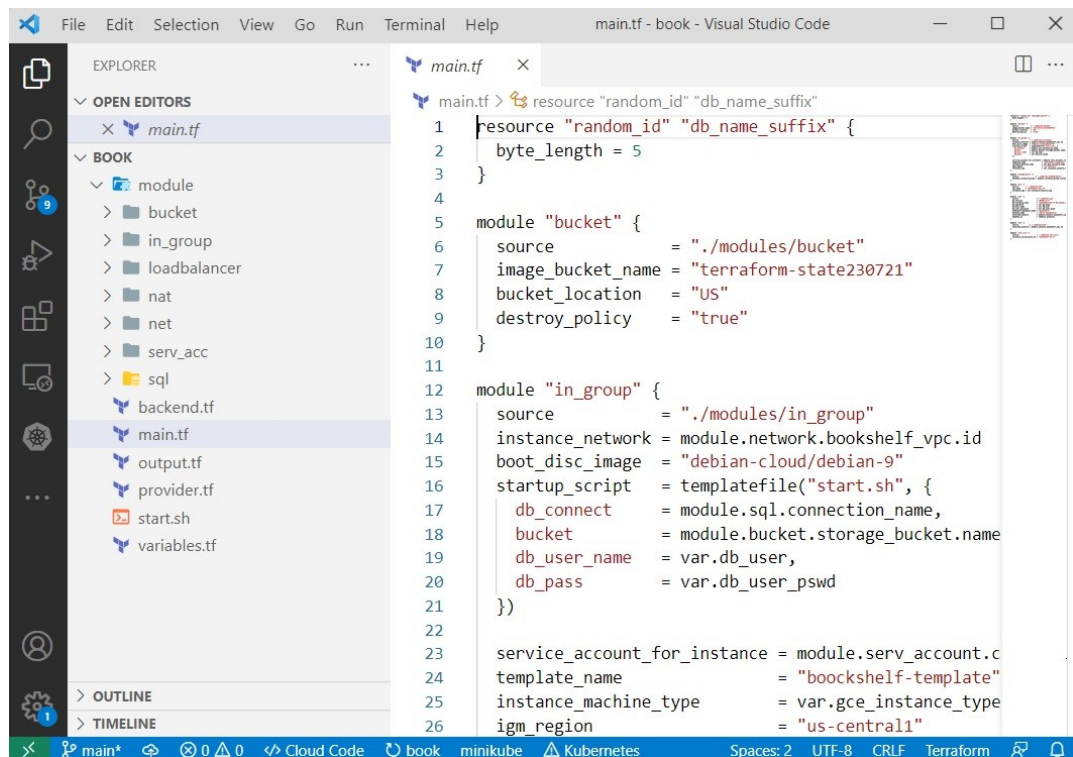
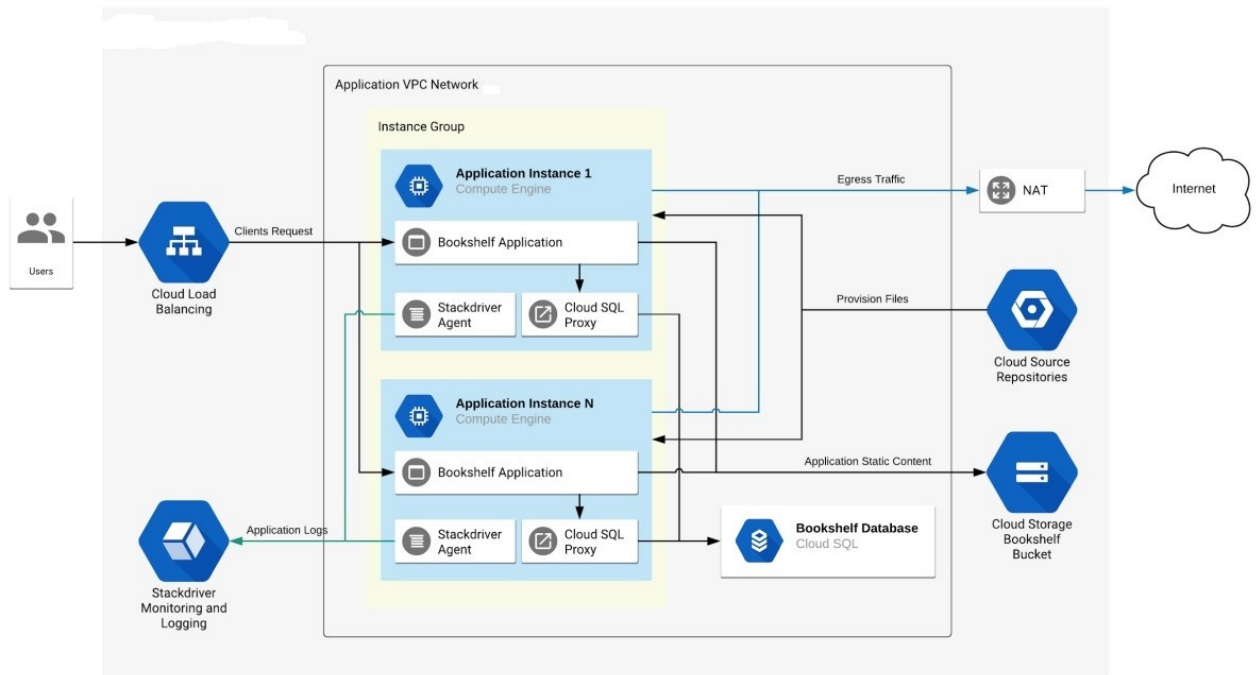


рисунок 3.1 — Visual Studio Code

3.2 Проектування і реалізація структури баз даних

Робоча схема проекту зображена на рисунку 3.2.



рисунк 3.2 — Схема роботи аплікації

В проєкті використано два типи баз даних — це Bookshelf Database(Cloud SQL), де зберігається інформація про користувачів та книжки та Cloud Storage Bookshelf Bucket, в якому зберігаються обкладинки книг у вигляді файлів JPEG.

Базовий меседж щодо використання Cloud SQL від Google є твердженням, що платформа керує екземпляром БД а користувач вже керує своїми даними.

Як варіанти використання надання хмарної альтернативи локальним базам даних MySQL, PostgreSQL та SQL Server. Менше часу на керування своєю базою даних та більше часу на її використання. Багато програм, що працюють на Compute Engine, App Engine та інших службах, використовують Cloud SQL для

зберігання баз даних. GUI надзвичайно зручний, створити БД займає часу 5 хвилин, дозволяючи працювати паралельно.

Інтерфейс Google Cloud Platform для роботи з SQL, рисунок 3.3.

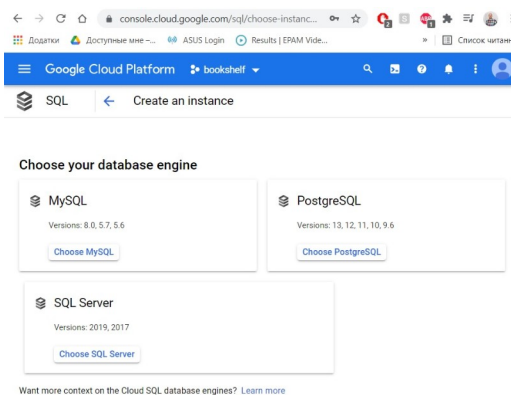


рисунок 3.3 — Інтерфейс Google Cloud Platform для роботи з SQL

Вибираємо потрібну базу даних, рисунок 3.4.

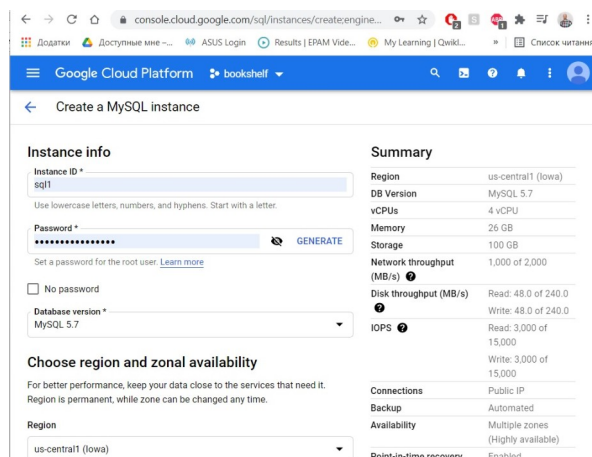


рисунок 3.4 — Створення бази даних

Платформа надає широкі можливості щодо кастомізації бази даних, рисунок 3.5.

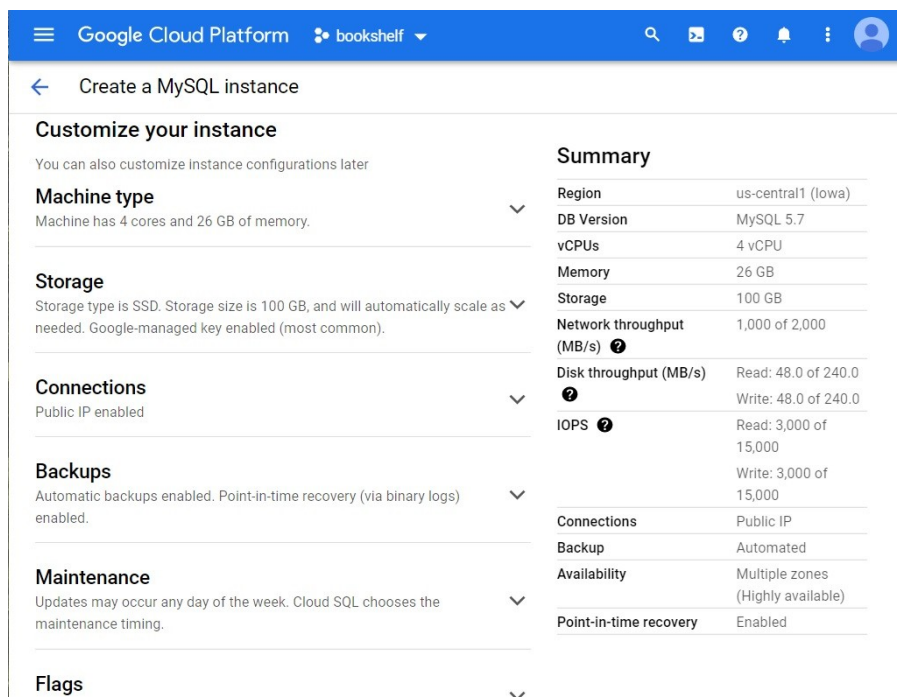


рисунок 3.5 — Можливості налаштувань бази даних

Базу даних створено. Необхідність роботи з GUI найчастіше виникає в умовах обмеженого використання, наприклад, інфраструктура створюється з цілями навчання. Під'єднуємося до бази за допомогою Cloud Shell, рисунок 3.6

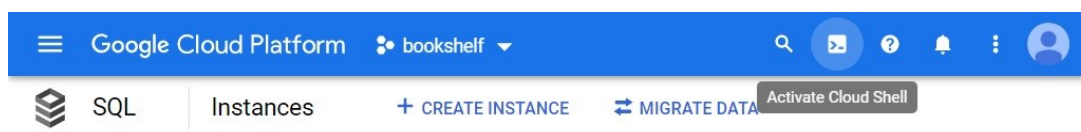


рисунок 3.6 — Під'єднання до бази даних за допомогою Cloud Shell

Вводимо потрібну командою `gcloud sql connect myinstance --user=root`

та починаємо працювати з базою даних.

Реалізація бази даних Cloud SQL в Terraform виглядає наступним чином, рисунок 3.7.

```
cloud_august2021@z1:~/book/one/modules/sql$ cat main.tf
resource "google_sql_database_instance" "bookshelf_db" {
  name                = var.db-instance_name
  database_version    = var.db_version
  root_password       = var.db_password
  region              = var.sql_region
  deletion_protection = false
  settings {
    tier = var.machine_type
    ip_configuration {
      ipv4_enabled = false
      private_network = var.selected_network
    }
  }
}

resource "google_sql_database" "bookshelf_db" {
  name       = var.bookshelf_database_name
  instance   = google_sql_database_instance.bookshelf_db.id
}

resource "google_sql_user" "users" {
  name       = var.db_user_name
  instance   = google_sql_database_instance.bookshelf_db.name
  password   = var.db_user_password
}
```

рисунок 3.7 — Код Terraform для створення бази даних SQL

Cloud Storage файли хмарного сховища впорядковані в бакети(buckets). Коли ми створюємо бакет то надаємо йому глобально унікальну назву з вказанням географічного місця, де він зберігається. Існує кілька способів контролю доступу користувачів до бакетів. Для більшості цілей достатньо Cloud IAM(Identity and

Access Management), але якщо потрібен більш точний контроль, можна створити списки контролю доступу, які пропонують більш точний контроль та визначають, хто має доступ до бакетів та об'єктів, а також який у них рівень доступу. Об'єкти Cloud Storage незмінні але можливо ввімкнути редагування об'єктів у бакетах. Історія змін, або перезапису чи видалення усіх об'єктів у бакеті, зберігається. Cloud Storage також пропонує політику керування життєвим циклом.

Бакет має класи зберігання для будь-якого навантаження, дозволяючи заощаджувати витрати, зберігаючи дані в різних класах зберігання.

1) Standard Storage. Підходить для даних до яких часто звертаються, включаючи веб-сайти, потокове відео та мобільні програми.

2) Nearline Storage. Низька вартість. Підходить для даних, які можна зберігати щонайменше 30 днів.

3) Coldline Storage. Дуже низька вартість. Підходить для даних, які можна зберігати щонайменше 90 днів, включаючи відновлення після збоїв.

4) Archive Storage. Найнижча вартість. Підходить для даних, які можна зберігати принаймні 365 днів, включаючи архіви нормативних документів.

При створенні вручну в Google Cloud Console йдемо до вкладки Cloud Storage Browser, рисунок 3.8.

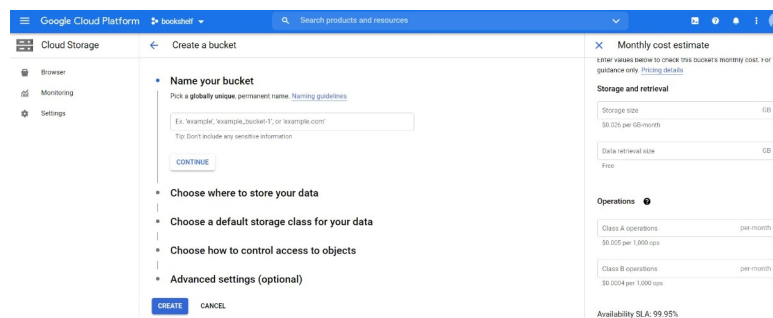


рисунок 3.8 — Налаштування параметрів сховища

Реалізація бази даних Cloud Storage в Terraform, рисунок 3.9

```
cloud_august2021@z1:~/book/one/modules/bucket$ cat main.tf
resource "google_storage_bucket" "image_storage" {
  name          = var.image_bucket_name
  location      = var.bucket_location
  force_destroy = var.destroy_policy
}

resource "google_storage_bucket_iam_member" "allusers" {
  bucket = google_storage_bucket.image_storage.name
  role   = "roles/storage.objectViewer"
  member = "allUsers"
}
cloud_august2021@z1:~/book/one/modules/bucket$
```

рисунок 3.9 — Код Terraform для створення бакета

3.3 Розробка функціоналу програмної реалізації для управління ресурсами гетерогенних БД

Структура файлів проекту представлена на рисунку 3.10.

```
cloud_august2021@z1:~/book/one$ tree
.
├── backend.tf
├── main.tf
├── modules
│   ├── bucket
│   │   ├── main.tf
│   │   ├── output.tf
│   │   └── variables.tf
│   ├── instance_group
│   │   ├── main.tf
│   │   ├── output.tf
│   │   └── variables.tf
│   ├── loadbalancer
│   │   ├── main.tf
│   │   ├── output.tf
│   │   └── variables.tf
│   ├── nat
│   │   ├── main.tf
│   │   └── variables.tf
│   ├── network
│   │   ├── main.tf
│   │   ├── output.tf
│   │   └── variables.tf
│   ├── serv_account
│   │   ├── main.tf
│   │   ├── output.tf
│   │   └── variables.tf
│   └── sql
│       ├── main.tf
│       ├── output.tf
│       └── variables.tf
├── output.tf
├── provider.tf
├── startup-script.sh
├── terraform.tfvars
└── variables.tf

8 directories, 27 files
cloud_august2021@z1:~/book/one$
```

рисунок 3.10 — структура Terraform коду проекту

Щоб створити інфраструктуру ми маємо в запусити три команди:

1. `terraform init`. Ця команда використовується для ініціалізації робочого каталогу, що містить файли конфігурації Terraform. Це перша команда, яку слід виконати після написання нової конфігурації Terraform або клонування вже існуючої з системи контролю версій. Команда ідемпотентна.

2. `terraform plan` Ця команда створює план виконання(execution plan). За замовчуванням створення плану складається з зчитування поточного стану будь-яких вже існуючих віддалених об'єктів щоб переконатися що стан Terraform є актуальним та наступне порівняння поточної конфігурації з попереднім станом з вказанням будь-яких відмінностей. Можливо скористатися цією командою щоб перевірити чи запропоновані зміни відповідають очікуванням, перш ніж застосувати ці зміни.

3. `terraform apply` Ця команда виконує дії, запропоновані в командою `terraform plan`. Найпростіший спосіб використання `terraform apply` – запустити його без будь-яких аргументів, і в цьому випадку він автоматично створить новий план виконання (ніби ми вже запускали `terraform plan`), а потім запропонує затвердити цей план перш ніж приймати вказані дії.

Terraform бере за основу `main.tf` і починає виконувати [16].

```
resource "random_id" "db_name_suffix" {
  byte_length = 5
}
```

```
module "bucket" {
  source      = "../modules/bucket"
```



```

image_bucket_name = "terraform-state-august2021"

bucket_location  = "EU"

destroy_policy   = "true"
}

```

```

module "instance_group" {

    source      = "./modules/instance_group"

    instance_network = module.network.bookshelf_vpc.id

    boot_disc_image = "debian-cloud/debian-9"

    startup_script = templatefile("startup-script.sh", {

        db_connect  = module.sql.connection_name,

        bucket      = module.bucket.storage_bucket.name,

        project_id  = var.project_id

        db_user_name = var.db_user,

        db_pass      = var.db_user_pswd

    })

```

```

                                service_account_for_instance      =
module.serv_account.custom_instance_service_account

    template_name      = "boockshelf-template"

    instance_machine_type = var.gce_instance_type

    igm_region          = "europe-west3"

```

```
instance_tag      = var.instance_network_tag
}
```

```
module "loadbalancer" {

  source      = "./modules/loadbalancer"

  backend_instance_group = module.instance_group.instance_grp
}
```

```
module "network" {

  source      = "./modules/network"

  vpc_name    = "bookshelf-vpc-tf"

  instance_tag = var.instance_network_tag
}
```

```
module "sql" {

  source      = "./modules/sql"

  db_version    = "MYSQL_5_7"

  db-instance_name    = "bookshelf-db-tf-${random_id.db_name_suffix.hex}"

  db_password        = var.db_pswd

  db_user_name       = var.db_user
}
```

```

db_user_password    = var.db_user_pswd

sql_region          = var.sql_region

bookshelf_database_name = "bookshelf"

machine_type        = "db-n1-standard-2"

selected_network     = module.network.bookshelf_vpc.id

depends_on            = [module.network]
}

```

```

module "nat" {

  source      = "../modules/nat"

  selected_network = module.network.bookshelf_vpc.id
}

```

```

module "serv_account" {

  source      = "../modules/serv_account"

  instance_servaccount_id = "bookshelf-sa-1"
}

```

модуль sql відповідає за створення бази даних SQL

```

resource "google_sql_database_instance" "bookshelf_db" {

  name          = var.db-instance_name

  database_version = var.db_version

  root_password    = var.db_password

  region          = var.sql_region

  deletion_protection = false

  settings {

    tier = var.machine_type

    ip_configuration {

      ipv4_enabled = false

      private_network = var.selected_network

    }

  }

}

```

```

resource "google_sql_database" "bookshelf_db" {

  name = var.bookshelf_database_name

  instance = google_sql_database_instance.bookshelf_db.id

}

```

```
resource "google_sql_user" "users" {
  name      = var.db_user_name
  instance = google_sql_database_instance.bookshelf_db.name
  password = var.db_user_password
}
```

модуль bucket відповідає за створення бази даних Cloud Storage

```
resource "google_storage_bucket" "image_storage" {
  name      = var.image_bucket_name
  location  = var.bucket_location
  force_destroy = var.destroy_policy
}
```

```
resource "google_storage_bucket_iam_member" "allusers" {
  bucket = google_storage_bucket.image_storage.name
  role   = "roles/storage.objectViewer"
  member = "allUsers"
```

3.4 Робота з інтерфейсом програмної реалізації для управління ресурсами гетерогенних БД

Після першого запуску аплікації ми потрапляємо на головну сторінку, рисунок 3.11.

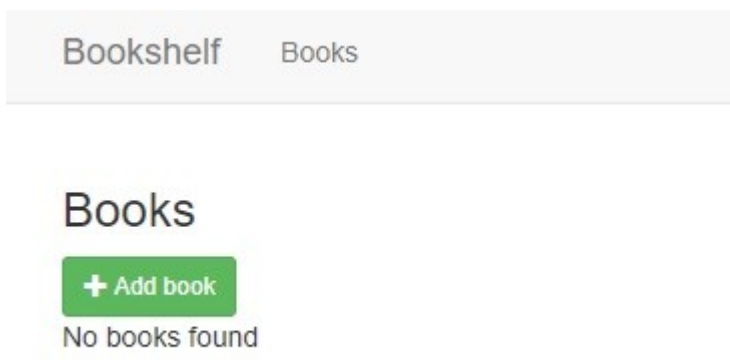


рисунок 3.11 — Початкова сторінка аплікації Bookshelf

Клікаємо на кнопку “+Add book” та додаємо потрібну книжку, рисунок 3.12.

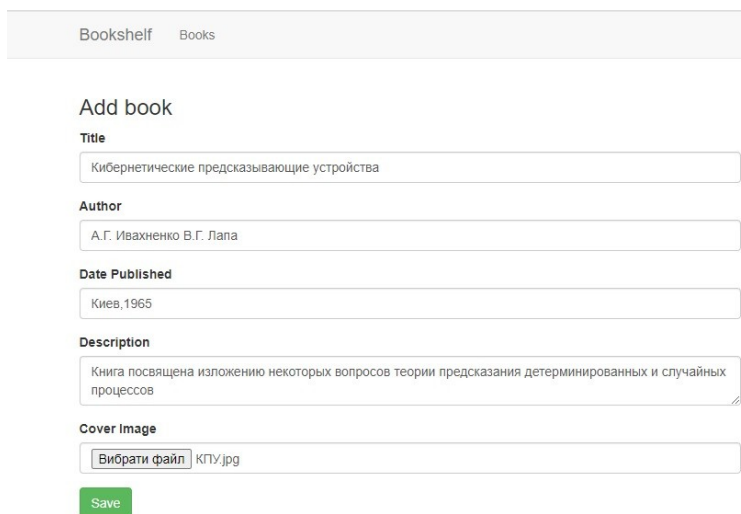


рисунок 3.12 — Додавання книги до Bookshelf

Натискаємо кнопку “save” додаючи книгу до баз даних, рисунок 3.13.



рисунок 3.13 — Зберігання книги в апікації Bookshelf

Для редагування або видалення інформації потрібно клікнути на обкладинку, рисунок 3.14.

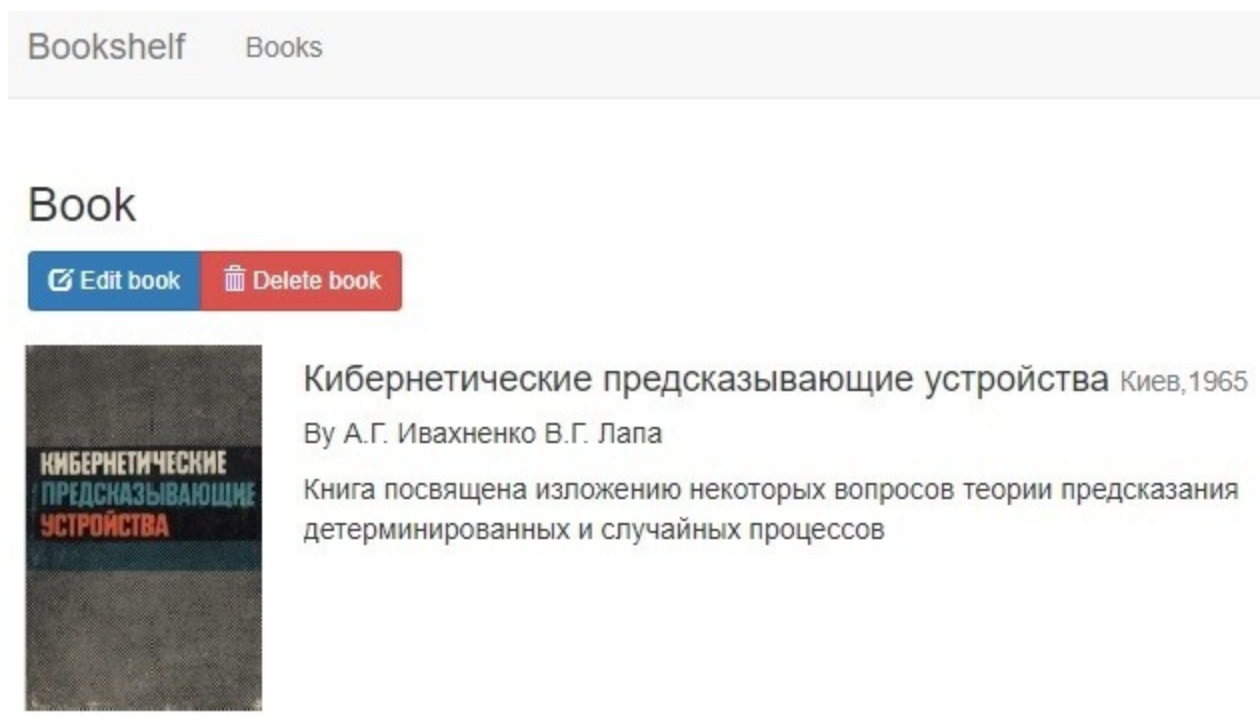


рисунок 3.14 — Возможности редагування аплікації Bookshelf

3.5 Висновки

У цьому розділі було проведено аналіз засобів для розробки програмних рішень моделі управління ресурсами баз даних та обраного варіанту для реалізації програмного продукту. Також було спроектовано бази даних, розроблені функціональні та інтерфейсні програми для спрощення використання системи.

У результаті було отримано робочий програмний продукт, який готовий до використання у відповідності до поставленого завдання. Програма була реалізована за допомогою Terraform та Bash на Google Cloud Platform. В майбутньому програмний продукт можливо значно покращити шляхом впровадження інструментів автоматизації, таких як Ansible, Chef, Puppet, Saltstack.

4 ФУНКЦІОНАЛЬНО-ВАРТІСТНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Постановка завдання

У даному розділі проведена оцінка основних функціональних та вартісних характеристик програмного продукту, розробленого для реалізації моделі управління ресурсами гетерогенних баз даних в хмарному середовищі.

Програмний продукти призначено для використання на персональних комп'ютерах під управлінням будь-якої операційної системи, оскільки веб-застосунок не залежить від технології реалізації апаратного забезпечення та операційної системи. Нижче наведено аналіз різних варіантів реалізації з метою вибору оптимальної стратегії створення програмного продукту, враховуючи при цьому як економічні фактори так і технічні фактори, для чого було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз – це технологія оцінки реальної вартості продукту або послуги незалежно від організаційної структури компанії. Прямі та побічні витрати розподіляються по продуктам та послугам у залежності від потрібних обсягів ресурсів на кожному етапі виробництва. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні найбільш оптимального розподілу ресурсів, що виділені на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Метод ФВА починається з визначення послідовності функцій, необхідних для виробництва продукту. Спочатку йдуть всі можливі функції, які розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не

впливають. Також на цьому етапі оптимізується сама послідовність скороченням кроків, що не впливають на витрати.

Для кожної функції визначається повний обсяг річних витрат та кількість робочих часів. На основі даних оцінок визначається кількісна характеристика джерел витрат. Після визначення джерел витрат проводиться кінцевий розрахунок витрат на виробництво продукту.

4.2 Обґрунтування функцій та параметрів дослідження

Метод ФВА був застосований для проведення техніко-економічний аналізу розробки моделі управління ресурсами гетерогенних баз даних в хмарному середовищі. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту.

До продукту були визначені наступні технічні вимоги:

- 1) можливість виконання на персональних комп'ютерах із стандартною конфігурацією;
- 2) висока швидкість обробки даних та відповідь користувачеві у реальному часі;
- 3) зручність та простота взаємодії;
- 4) можливість зручного налаштування, масштабування та обслуговування;
- 5) мінімальні витрати на впровадження програмного продукту.

Головна функція F_0 – розробка програмного продукту, який вирішує задачу моделі управління ресурсами гетерогенних баз даних в хмарному середовищі та будує відповідну модель НМ.

Виходячи з конкретної мети, можна виділити наступні основні функції програмного продукту:

- F_1 – вибір постачальника хмарних послуг;
- F_2 – вибір інструмента IaC;
- F_3 – вибір середовища розробки.

Кожна з основних функцій може мати декілька варіантів реалізації, які представлені в таблиці 4.1

Таблиця 4.1 — Варіанти реалізацій функцій

Варіант реалізації/функції	Функція F_1	Функція F_2	Функція F_3
Варіант реалізації А	AWS	Terraform	PyCharm
Варіант реалізації Б	GCP	Chef	VSCode
Варіант реалізації С	Azure	Ansible	-
Варіант реалізації Д	-	Puppet	-
Варіант реалізації Е	-	Saltstack	-

Варіанти реалізації основних функцій наведені у морфологічній карті системи яка відображає всі можливі комбінації на рисунку 4.1.

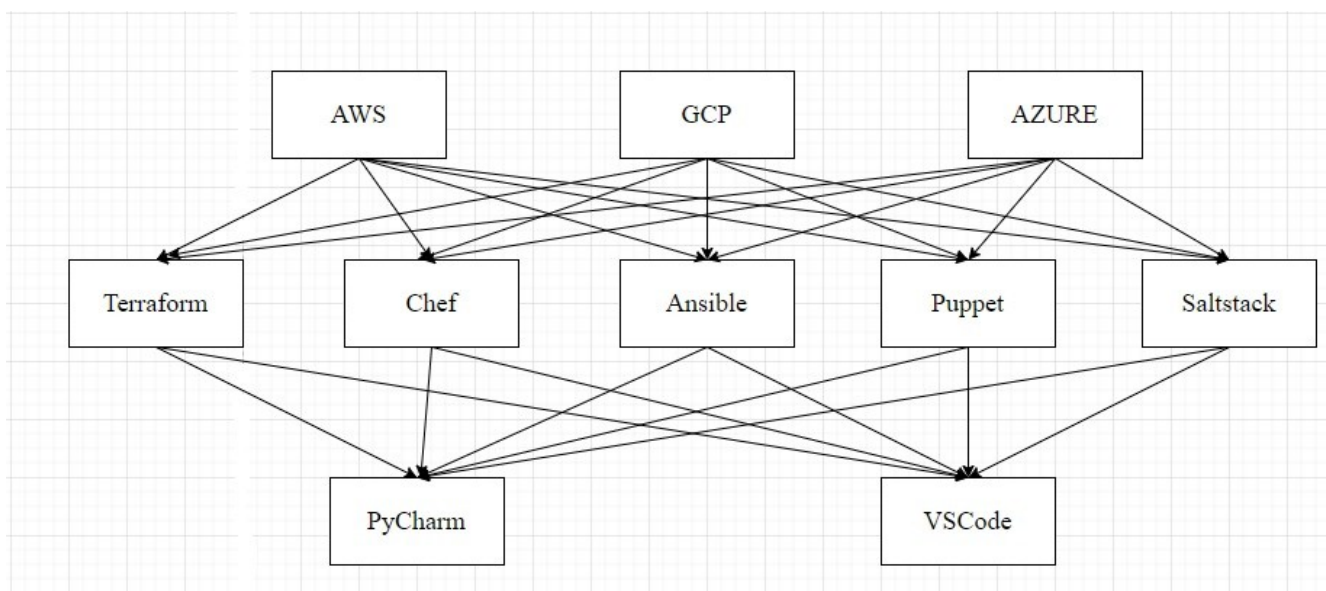


Рисунок 4.1 — Морфологічна карта варіантів реалізації функцій

На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій, як у таблиці 4.2.

Таблиця 4.2 — Позитивно-негативна матриця варіантів основних функцій

Основні функції	Варіанти реалізації	Переваги	Недоліки
F ₁	А	Наявність туторіалів та практичних прикладів реалізації функціоналу	Складність реалізації потрібного функціоналу
	Б	Інтуїтивно зрозумілий інтерфейс та ґрунтовна офіційна документація	Можуть знадобитися додаткові аккаунти
	С	Широкий вибір вузькоспеціалізованих рішень	Слабка візуалізація інтерфейсу
F ₂	А	Широкий вибір плагінів для реалізації функціоналу	Потребує окремої шифровки файлу кінцевого стану
	Б	Спеціалізований інструмент саме для автоматизації IaC	Процедурна мова імплементації
	С	Якісна документація, великий вибір плагінів, використання YAML	Складна процедура зміни конфігурації інфраструктури
	Д	Процедурна мова імплементації	Питання з безпекою використання
	Е	Активна підтримка спільноти, використання мови програмування Python	Імплементація рішення займає занадто багато часу
F ₃	А	Зручна у використанні, активна ком'юніті	Все найкраще в платній версії
	Б	Безкоштовний інструмент з великим вибором плагінів	Займає багато пам'яті на диску

На основі аналізу позитивно-негативної матриці робимо висновок про доцільність відбракування певних варіантів, оскільки вони не відповідають поставленим перед програмним продуктом задачам.

Функція F_1 : оскільки вибір хмарного провайдера для вирішення поставленої задачі є важливим в контексті зниженні затрат на розробку та підтримання інфраструктури, необхідно обрати провайдера с найбільш прийнятними умовами роботи та обслуговування.

Функція F_2 : оскільки вибір інструмента для імплементації інфраструктури для вирішення поставленої задачі є часткова взаємозамінним, можемо розглянути два інструмента.

Функція F_3 : оскільки один з інструментів можливо використати у нашій роботі виключно у платній версії, ми її виключаємо.

Прийнятні варіанти:

- $F_1(B) \text{ — } F_2(A) \text{ — } F_3(B)$
- $F_1(B) \text{ — } F_2(C) \text{ — } F_3(B)$

Для оцінювання якості розглянутих функцій обраємо систему параметрів.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X_1 – час на вивчення теоретичної основи;
- X_2 – час на вивчення можливостей вибраних інструментів;
- X_3 – час на проектування системи;
- X_4 – час на імплементацію програмних рішень.

Для F_1 застосуємо параметри X_1 та X_2 , для F_2 застосуємо параметри X_2 , X_3 , X_4 для F_3 застосуємо параметри X_2 , X_4

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту як показано у таблиці 4.3.

Таблиця 4.3 — Основні параметри програмного продукту

Умовні позначення	Одиниці виміру	Значення параметрів		
		Гірші	Середні	Кращі
X_1	години	45	20	12
X_2	години	35	20	10
X_3	години	40	30	15
X_4	години	45	25	15
X_5	години	55	30	15

За даними таблиці 4.3 будемо графічні характеристики параметрів, рисунок 4.3

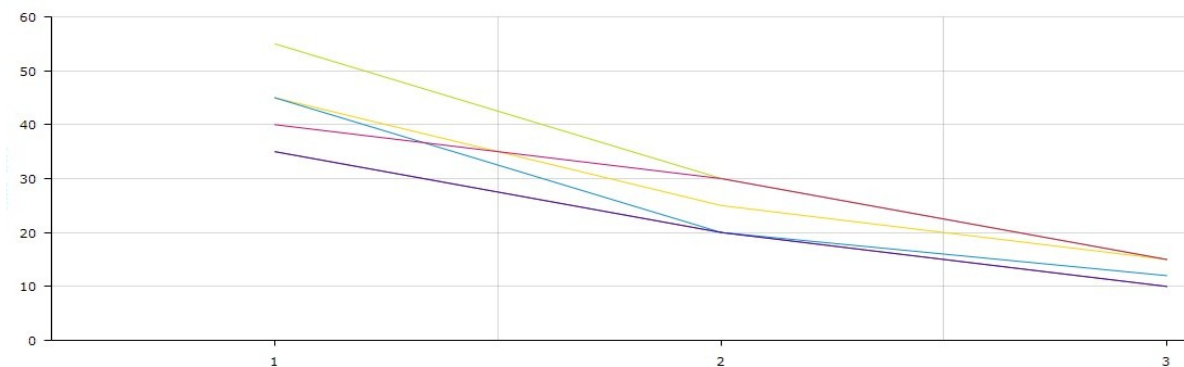


Рисунок 4.3 – Графічні характеристики параметрів

4.3 Визначення коефіцієнтів значимості параметрів

Значимість кожного параметра визначається методом попарного порівняння на основі результатів ранжування експертами та попарного порівняння параметрів.

Таблиця 4.4 — Результати ранжування показників

Умовне позначення	Одиниці вимірювання	Ранг параметра за оцінкою експерта							Сума рангів в R_i	Відхилення Δ_i	Δ_i^2
		1	2	3	4	5	6	7			
X_1	Год.	5	5	4	3	4	5	4	30	10	100
X_2	Год.	2	5	3	3	3	4	4	24	9	81
X_3	Год.	2	1	2	4	3	1	1	14	-9	81
X_4	Год.	2	3	3	3	4	4	2	21	8	64
X_5	Год.	4	1	3	2	1	1	4	16	-4	16
Разом		15	15	15	15	15	15	15	105	0	342

Для перевірки ступеню достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^N r_{ij} = 50,$$

де r_{ij} – ранг i -го параметра, визначений j -м експертом;
 N – число експертів.б) середня сума рангів T :

$$T = \frac{1}{n} R_i = 12,5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T.$$

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^n \Delta_i^2 = 125$$

д) коефіцієнт узгодженості (конкордації):

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 242}{7^2(5^3 - 5)} = 1 > W_k = 0,67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67. Скориставшись результатами ранжування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.5. Більш ваговим є коефіцієнт 1. Числове значення, що визначає ступінь переваги i -го параметра над j -тим, a_{ij} визначається за формулою:

$$a_{ij} = \begin{cases} 1,5 & x_i > x_j; \\ 1,0 & x_i = x_j; \\ 0,5 & x_i < x_j. \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A=\|a_{ij}\|$. Для кожного параметра розрахунок вагомості K_{B_i} проводиться за наступною формулою:

$$K_{B_i} = \frac{b_i}{\sum_{i=1}^n b_i},$$

де $b_i = \sum_{j=1}^N a_{ij}$ – вагомість i -го параметра за результатами оцінок всіх експертів;

a_{ij} – коефіцієнт переваги i -го на j -тим параметром.

Таблиця 4.5 — Результати ранжування параметрів

Параметри	Експерти							Підсумкова оцінка	Числове значення коефіцієнтів переваги
	1	2	3	4	5	6	7		
X1,X2	>	>	>	>	>	<	>	>	1.5
X1,X3	>	>	<	>	>	<	>	>	1.5
X1,X4	>	>	>	<	>	<	>	>	1.5
X1,X5	>	>	<	<	<	>	>	>	1.5
X2,X3	>	>	>	>	>	<	>	>	1.5
X2,X4	>	<	<	>	>	>	>	>	1.5
X2,X5	<	<	>	<	>	<	<	<	0.5
X3,X4	>	>	>	>	>	<	>	>	1.5
X3,X5	>	>	<	<	<	<	>	<	0.5
X4,X5	>	>	>	>	>	<	>	>	1.5

Таблиця 4.6 — Результати ранжування параметрів

X_i	X_j					Перша ітерація		Друга ітерація		Третя ітерація	
	X_1	X_2	X_3	X_4	X_5	b_i	K_{bi}	b_i^2	K_{bi}^2	b_i^3	K_{bi}^3
X_1	1	1.5	1.5	1.5	1.5	6	0.24	32	0.239	124.75	0.239
X_2	0.5	1	0.5	1.5	1.5	5	0.2	22.5	0.191	104	0.19
X_3	0.5	0.5	1	0.5	1.5	6	0.24	32	0.239	124.75	0.239
X_4	1.5	1.5	0.5	1	1.5	4	0.16	16.5	0.152	80.25	0.15
X_5	0.5	0.5	0.5	1.5	1	3	0.12	14	0.122	64.5	0.123
Разом						24	1	117	1	498	1

4.4 Аналіз рівня якості варіантів реалізації функцій програмного продукту

Таблиця 4.7 – Розрахунок показників якості

Основні функції	Варіант реалізації	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F ₁	В	X ₁	45	7	0.239	1.673
		X ₂	35	6	0.19	1.14
F ₂	А	X ₂	35	8	0.19	1.52
		X ₃	40	6	0.239	1.434
		X ₄	45	4	0.15	0.6
F ₂	С	X ₂	35	8	0.19	1.52
		X ₃	40	6	0.239	1.434
		X ₄	45	3	0.15	0.45
F ₃	В	X ₂	35	8	0.19	1.52
		X ₄	45	2	0.15	0.3

Визначимо коефіцієнт технічного рівня кожного з варіантів:

$$K_{K1}=1.673+1.14+1.52+1.434+0.6+1.52+0.3=8.187$$

$$K_{K2}=1.673+1.14+1.52+1.434+0.45+1.52+0.3=8.037$$

Найкращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.5 Економічний аналіз варіантів розробки програмного продукту

Загальна трудомісткість обчислюється за формулою.

$$T_o = T_p \cdot K_{\Pi} \cdot K_{CK} \cdot K_M \cdot K_{CT} \cdot K_{CT.M},$$

де T_p – трудомісткість розробки програмного продукту;
 K_{Π} – поправочний коефіцієнт;
 K_{CK} – коефіцієнт на складність вхідної інформації;
 K_M – коефіцієнт рівня мови програмування;
 K_{CT} – коефіцієнт використання стандартних модулів і прикладних програм;
 $K_{CT.M}$ – коефіцієнт стандартного математичного забезпечення.

Для визначення вартості розробки програмного продукту спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Проектування та розробка проекту;
2. Розробка баз даних;
3. Робота з базою даних.

Завдання 1 та 2 за ступенем відносимо до групи складності 1, завдання 3 – до групи складності 3.

Для задач 1 і 2, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює:

$T_p = 90$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого і другого завдання: $K_{\Pi} = 1,7$.

Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $K_{CK} = 1$. Оскільки при розробці першого і

другого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{ст} = 0,8$. Тоді загальна трудомісткість виконання першого і другого завдання дорівнює:

$$T_1 = 90 * 1,7 * 0,8 = 122,4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для задачі 3. Виходячи із норм часу для завдань проєктувального характеру ступеня новизни Б та групи складності 3, трудомісткість дорівнює:

$T_p = 70$ людино-днів, $K_n = 0,9$, $K_{ск} = 1$, $K_{ст} = 0,8$. Тоді загальна трудомісткість виконання третього завдання дорівнює:

$$T_1 = 70 * 0,9 * 0,8 = 50,4 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань:

$$T_1 = (122,4 + 122,4 + 50,4) \cdot 8 = 2361,6 \text{ людино-годин.}$$

Виходячи з вище проведених розрахунків, можна зробити висновок, що найбільш трудомістким завданням є перше і друге завдання. Далі вважається, що робочий день складає 8 годин, в тиждні п'ять робочих днів. В розробці беруть участь двоє програмістів. Причому один з них займається розробкою функціоналу програмного продукту та має оклад в 60000 грн, а один – розробкою баз даних та має оклад в 50000 грн. Також до роботи залучено тестувальника з окладом у 35000 грн.

Визначимо середню заробітну плату за годину за формулою:

$$C_q = \frac{M}{T_m \cdot t} \text{ грн.},$$

де M – місячний оклад працівників;
 T_m – кількість робочих днів на місяць;
 t – кількість робочих годин в день.

$$C_q = \frac{60000 + 50000 + 35000}{3 \cdot 20 \cdot 8} = 302.08 \text{ грн}$$

Тоді, розрахуємо заробітну плату за формулою:

$$C_{зп} = C_q \cdot T_i \cdot K_d,$$

де C_q – величина погодинної оплати праці програміста;
 T_i – трудомісткість відповідного завдання;
 K_d – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$C_{зп} = 302.08 \cdot 8 \cdot 2362 = 5708103.68 \text{ грн.}$$

Відрахування на соціальний внесок становить 22%:

$$C_{св} = C_{зп} \cdot 0,22 = 5708103.68 \cdot 0,22 = 1255782.81 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. Оскільки одна ЕОМ обслуговується одним інженером апаратного забезпечення з окладом 50000 грн. та коефіцієнтом зайнятості $K_3 = 0,2$ то для однієї машини отримаємо:

$$C_r = 12 \cdot M \cdot K_3 = 12 \cdot 50\,000 \cdot 0,2 = 120\,000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{3П} = C_{Г} \cdot (1 + K_3) = 120\,000 \cdot (1 + 0,2) = 122\,400 \text{ грн.}$$

Відрахування на соціальний внесок становить 22%:

$$C_{СВ} = C_{3П} \cdot 0,22 = 122\,400 \cdot 0,22 = 26\,928 \text{ грн.}$$

Амортизаційні відрахування розраховуємо за формулою при амортизації 25% та вартості ЕОМ – 20 000 грн.:

$$C_A = K_{ТМ} \cdot K_A \cdot Ц_{ПР} = 1,15 \cdot 0,25 \cdot 20\,000 = 5\,750 \text{ грн.}$$

де $K_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

K_A – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо за формулою:

$$C_P = K_{ТМ} \cdot K_P \cdot Ц_{ПР} = 1,15 \cdot 0,05 \cdot 20\,000 = 1\,150 \text{ грн.}$$

де K_P – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{ЕФ} = (Д_K - Д_B - Д_C - Д_P) \cdot t \cdot K_B, T_{ЕФ} = (365 - 104 - 12 - 16) \cdot 8 \cdot 0,9 = 1\,612,08 \text{ год.}$$

де $Д_K$ – календарна кількість днів у році;

$Д_B, Д_C$ – відповідно кількість вихідних та святкових днів;

$Д_P$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{EL} = T_{EF} \cdot N_C \cdot K_3 \cdot \Pi_{EL} = 1612,8 \cdot 0,56 \cdot 0,2 \cdot 2,92 = 527,45 \text{ грн.}$$

де N_C – середньо-споживча потужність приладу;

K_3 – коефіцієнтом зайнятості приладу;

Π_{EL} – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_H = \Pi_{PP} \cdot 0,67 = 20\,000 \cdot 0,67 = 13\,400 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть складати:

$$C_{EK} = C_{3П} + C_{CB} + C_A + C_P + C_{EL} + C_H,$$

$$C_{EK} = 122\,400 + 6\,336 + 5\,750 + 1\,150 + 527,45 + 13\,400 = 149\,563,45 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{MG} = \frac{C_{EK}}{T_{EF}} = \frac{149\,563,45}{1612,8} = 92,74 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу складають:

$$C_M = C_{MG} \cdot T = 92,74 \cdot 1134,72 = 105\,234 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{3П} \cdot 0,67 = 122\,400 \cdot 0,67 = 82\,008 \text{ грн.}$$

Отже, вартість розробки програмного продукту за варіантами становить:

$$C_{ПП} = C_{3П} + C_{СВ} + C_M + C_H, C_{ПП} = 122\,400 + 26\,928 + 105\,234 + 82\,008 = 336\,570 \text{ грн.}$$

4.6 Вибір кращого варіанту програмного продукту техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{TEP1} = \frac{K_K}{C_{ПП}} = \frac{8,187}{336\,570} = 2,432 \cdot 10^{-5}$$

$$K_{TEP2} = \frac{K_K}{C_{ПП}} = \frac{8,037}{336\,570} = 2,39 \cdot 10^{-5}$$

Оскільки $K_{TEP1} = 2,432 \cdot 10^{-5}$ має більше значення, то більш ефективним є перший варіант.

Отже, за результатом проведеного функціонально-вартісного аналізу можемо зробити висновок, що з альтернатив, які залишилися після відбору, доцільніше використовувати перший варіант реалізації продукту, а саме реалізація продукту на платформі Google Cloud Platform з використанням інструменту IaC Terraform на IDE VSCode. Таким чином, отримано універсальний підхід для реалізації моделі управління ресурсами гетерогенних баз даних в хмарному середовищі.

4.7 Висновки до розділу

В результаті виконання економічного розділу були систематизовані і закріплені теоретичні знання в галузі економіки та організації виробництва використанням їх для техніко-економічного обґрунтування розробки методом функціонально-вартісного аналізу. Були розглянуті два основні варіанти проведення аналізу: технічний аналіз продукту та дослідження варіантів на оптимальність з економічної точки зору.

В рамках технічного аналізу були сформульовані функції, які має виконувати програмний продукт, а також альтернативні варіанти їх реалізації. Для кожного з варіантів реалізації були обчислені основні параметри та виконане експертне оцінювання. Після отримання відповідних даних був обчислений коефіцієнт технічного рівня, завдяки якому відбувся вибір найкращої альтернативи.

Далі варіанти реалізації основних функцій програмного продукту були дослідженні на оптимальність з економічної точки зору. Вибір найкращої альтернативи здійснювався на основі коефіцієнту техніко-економічного рівня. Для кожної з досліджуваних альтернатив були обчислені такі параметри, як трудомісткість, витрати на трудові ресурси, витрати на матеріальне обладнання, накладні витрати, витрати на оплату машинного часу.

На основі даних про зміст основних функцій, які повинен реалізувати програмний продукт, були визначені два найбільш перспективні варіанти реалізації продукту. Найбільш ефективним виявився перший варіант реалізації, який варіант передбачає:

хмарна платформа — Google Cloud Platform;

інструмент створення інфраструктури — Terraform;

IDE VSCode.

ВИСНОВКИ

Метою роботи була розробка моделі управління ресурсами гетерогенних баз даних в хмарному середовищі Google Cloud Platform та підтримка її стабільної роботи з можливостями балансування навантаження та масштабування на випадок різких пікових навантажень.

В рамках імплементації завдання був ґрунтовно досліджений такий інструмент для створення та підтримки інфраструктури як Terraform, декларативний стиль програмування якого дозволяє надзвичайно швидко імплементувати ідеї щодо кількісних та якісних інфраструктурних параметрів. Питання може виникати вже на етапі подальшої підтримки існуючого функціоналу, оскільки це коштує достатньо дорого. Шлях подальшої ж модернізації додатка лежить в площині автоматизації розгортання самого додатку шляхом впровадження таких інструментів як Ansible, Chef, Puppet.

Запропонований в дипломній роботі проект моделі управління ресурсами гетерогенних баз даних в хмарному середовищі є життєздатним та простим продуктом але він потребує чималих коштів на утримання, які потрібно регулярно сплачувати провайдеру хмарних послуг.

В ході розробки моделі були описані бази даних, представлена модель роботи додатка, розроблено та описано функціонал. Розробка описаної моделі відбувалась на синтаксисі Terraform HCL(HashiCorp Configuration Language), системи управління базами Cloud SQL та Cloud Storage.

Застосування додатку можливе в рамках невеликого проекту або з навчальною метою.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Провайдер хмарних послуг Google Cloud Platform URL: <https://cloud.google.com/> (дата звернення: 26.08.2021)
2. Провайдер хмарних послуг Amazon Web Services URL: <https://aws.amazon.com/> (дата звернення: 26.08.2021)
3. Провайдер хмарних послуг Microsoft Azure Cloud Platform URL: <https://azure.microsoft.com/> (дата звернення: 26.08.2021)
4. Офіційний сайт інструменту Terraform by HashiCorp URL: <https://www.terraform.io/> (дата звернення: 26.08.2021)
5. Офіційний сайт інструменту Ansible URL: <https://www.ansible.com/> (дата звернення: 26.08.2021)
6. Офіційний сайт інструменту Puppet URL: <https://puppet.com/> (дата звернення: 26.08.2021)
7. Офіційний сайт інструменту Chef URL: <https://www.chef.io/> (дата звернення: 26.08.2021)
8. Офіційний сайт інструменту Saltstack URL: <https://saltproject.io/> (дата звернення: 26.08.2021)
9. Гайна Г. А. Основи проектування баз даних - Київ: Кондор. 2018. 19 с.
10. Robert C. Martin Clean Architecture: A Craftsman's Guide to Software Structure and Design. New York: Prentice Hall, 2017. 428 с.
11. Питер Б., Джоунс К., Петофф Д. Мерфи Н. Р., Site Reliability Engineering. Питер: O'REILLY 2018. 118 с.
12. Хоштейн Л., Мозер Р., Запускаем Ansible. Москва: O'REILLY 2018. 26 с.
13. Kim G., Behr K., Spafford G., The Phoenix Project. Portland, OR 97210 "IT Revolution". 2014. 374 с.

14. Kim G., Debois P., Humble J., Willis J., The DevOps Handbook: How to Create World-Class Agility, Reliability and Security in Technology Organizations. Portland, OR 97210 “IT Revolution” 2014. 198 c.
15. Brikman Y. Terraform: Up & Running. New York: O'REILLY, 2018. 115 c.
16. Bakhovskiy M., Terraform – the art of troubleshooting. Lviv: EPAM system, 2021. 100 c.
17. Erl T., Cope R., Naserpour A., Cloud Computing Design Patterns. New York: Prentice Hall, 2015. 403 c.
18. Limoncelli T., Chalup S., Hogan C., The Practice of Cloud System Administration Designing and Operating Large Distributed Systems. Boston: Addison-Wesley, 2018. 287 c.
19. Hernandez M., Database Design for Mere Mortals. New York: Prentice Hall, 2020. 287 c.
20. Churcher C., Beginning Database Design. New York: APress, 2017. 213 c.
21. Kleppmann M., Designing Data-Intensive Applications The Big Ideas Behind Reliable, Scalable and Maintainable Systems. New York: O'REILLY, 2017. 27 c.

Додаток А лістинг програми

```
main.tf
```

```
resource "random_id" "db_name_suffix" {  
  
  byte_length = 5  
  
}
```

```
module "bucket" {  
  
  source      = "./modules/bucket"  
  
  image_bucket_name = "terraform-state-august2021"  
  
  bucket_location = "EU"  
  
  destroy_policy = "true"  
  
}
```

```
module "instance_group" {  
  
  source      = "./modules/instance_group"  
  
  instance_network = module.network.bookshelf_vpc.id  
  
  boot_disc_image = "debian-cloud/debian-9"  
  
  startup_script = templatefile("startup-script.sh", {  
  
    db_connect = module.sql.connection_name,
```



```

    bucket      = module.bucket.storage_bucket.name,

    project_id   = var.project_id

    db_user_name = var.db_user,

    db_pass      = var.db_user_pswd

  })

  service_account_for_instance =
module.serv_account.custom_instance_service_account

  template_name      = "boockshelf-template"

  instance_machine_type    = var.gce_instance_type

  igm_region           = "europe-west3"

  instance_tag          = var.instance_network_tag
}

module "loadbalancer" {

  source              = "./modules/loadbalancer"

  backend_instance_group = module.instance_group.instance_grp
}

module "network" {

  source      = "./modules/network"

```

```

vpc_name    = "bookshelf-vpc-tf"

instance_tag = var.instance_network_tag
}

module "sql" {

    source          = "./modules/sql"

    db_version      = "MYSQL_5_7"

    db-instance_name    = "bookshelf-db-tf-${random_id.db_name_suffix.hex}"

    db_password        = var.db_pswd

    db_user_name        = var.db_user

    db_user_password    = var.db_user_pswd

    sql_region         = var.sql_region

    bookshelf_database_name = "bookshelf"

    machine_type        = "db-n1-standard-2"

    selected_network     = module.network.bookshelf_vpc.id

    depends_on          = [module.network]
}

module "nat" {

    source          = "./modules/nat"

    selected_network = module.network.bookshelf_vpc.id

```

```
}
```

```
module "serv_account" {  
  source      = "../modules/serv_account"  
  instance_servaccount_id = "bookshelf-sa-1"  
}
```

```
backend.tf  
terraform {  
  backend "gcs" {  
    bucket = "terraform-state080821"  
    prefix = "terraform/state"  
  }  
}
```

```
provider.tf  
provider "google" {  
  project = "bookshelf-322313"  
  region  = "us-central1"  
  zone    = "us-central1-a"
```

```
variables.tf
```

```
variable "db_user" {  
  
    sensitive = true  
  
}
```

```
variable "db_pswd" {  
  
    sensitive = true  
  
}
```

```
variable "db_user_pswd" {  
  
    sensitive = true  
  
}
```

```
variable "gce_instance_type" {}
```

```
variable "instance_network_tag" {}
```

```
variable "image_bucket_name" {}
```

```
variable "bucket_location" {}
```

```
variable "destroy_policy" {}
```

```
variable "instance_network" {}
```

```
variable "startup_script" {}
```

```
variable "service_account_for_instance" {}
```

```
variable "region" {}
```

```
variable "project_id" {}
```

```
variable "sql_region" {}
```

```
startup-script.sh
```

```
# Install Stackdriver logging agent
```

```
curl -sSO https://dl.google.com/cloudagents/install-logging-agent.sh
```

```
sudo bash install-logging-agent.sh
```

```
# Install or update needed software
```

```
apt-get update
```

```
apt-get install -yq git supervisor python python-pip
```

```
pip install --upgrade pip virtualenv
```

```
# Account to own server process
```

```
useradd -m -d /home/pythonapp pythonapp
```

```
# Fetch source code
```

```
export HOME=/root
```

```
git clone https://github.com/StanslavYermolenko/getting-started-python.git -b steps  
/opt/app
```

```
#cloud SQL
```

```
wget https://dl.google.com/cloudsql/cloud_sql_proxy.linux.amd64 -O cloud_sql_proxy
```

```
chmod +x cloud_sql_proxy
```

```
./cloud_sql_proxy -instances=${db_connect}=tcp:0.0.0.0:3306 &
```

```
cat <<EOF > /opt/app/7-gce/config.py
```

```
# See the License for the specific language governing permissions and
```

```
# limitations under the License.
```

```
"""
```

This file contains all of the configuration values for the application.

Update this file with the values for your specific Google Cloud project.

You can create and manage projects at <https://console.developers.google.com>

```
"""
```

```
import os
```

```
# The secret key is used by Flask to encrypt session cookies.
```

```
SECRET_KEY = 'secret'
```

```
# There are three different ways to store the data in the application.
```

```
# You can choose 'datastore', 'cloudsql', or 'mongodb'. Be sure to
```

```
# configure the respective settings for the one you choose below.
```

```
# You do not have to configure the other data backends. If unsure, choose
```

```
# 'datastore' as it does not require any additional configuration.
```

```
DATA_BACKEND = 'cloudsql'
```

```
# Google Cloud Project ID. This can be found on the 'Overview' page at
# https://console.developers.google.com
PROJECT_ID = "${project_id}"

# CloudSQL & SQLAlchemy configuration
# Replace the following values the respective values of your Cloud SQL
# instance.

CLOUDSQL_USER = "${db_user_name}"
CLOUDSQL_PASSWORD = "${db_pass}"
CLOUDSQL_DATABASE = "bookshelf"

# Set this value to the Cloud SQL connection name, e.g.
# "project:region:cloudsql-instance".
# You must also update the value in app.yaml.

CLOUDSQL_CONNECTION_NAME = "${db_connect}"

# The CloudSQL proxy is used locally to connect to the cloudsql instance.
# To start the proxy, use:
#
# $ cloud_sql_proxy -instances=your-connection-name=tcp:3306
#
```



```
# Port 3306 is the standard MySQL port. If you need to use a different port,
# change the 3306 to a different port number.
```

```
# Alternatively, you could use a local MySQL instance for testing.
```

```
LOCAL_SQLALCHEMY_DATABASE_URI = (
    'mysql+pymysql://{user}:{password}@127.0.0.1:3306/{database}').format(
        user=CLOUDSQL_USER, password=CLOUDSQL_PASSWORD,
        database=CLOUDSQL_DATABASE)
```

```
# When running on App Engine a unix socket is used to connect to the cloudsql
# instance.
```

```
LIVE_SQLALCHEMY_DATABASE_URI = (
    'mysql+pymysql://{user}:{password}@localhost/{database}'
    '?unix_socket=/cloudsql/{connection_name}').format(
        user=CLOUDSQL_USER, password=CLOUDSQL_PASSWORD,
        database=CLOUDSQL_DATABASE,
        connection_name=CLOUDSQL_CONNECTION_NAME)
if os.environ.get('GAE_INSTANCE'):
    SQLALCHEMY_DATABASE_URI = LIVE_SQLALCHEMY_DATABASE_URI
else:
```

```
SQLALCHEMY_DATABASE_URI = LOCAL_SQLALCHEMY_DATABASE_URI
```

```
# Mongo configuration
```

```
# If using mongolab, the connection URI is available from the mongolab control
```

```
# panel. If self-hosting on compute engine, replace the values below.
```

```
MONGO_URI = 'mongodb://user:password@host:27017/database'
```

```
# Google Cloud Storage and upload settings.
```

```
# Typically, you'll name your bucket the same as your project. To create a
```

```
# bucket:
```

```
#
```

```
# $ gsutil mb gs://<your-bucket-name>
```

```
#
```

```
# You also need to make sure that the default ACL is set to public-read,
```

```
# otherwise users will not be able to see their upload images:
```

```
#
```

```
# $ gsutil defacl set public-read gs://<your-bucket-name>
```

```
#
```

```
# You can adjust the max content length and allow extensions settings to allow
```

```
# larger or more varied file types if desired.
```

```
CLOUD_STORAGE_BUCKET = "${bucket}"
```

```
MAX_CONTENT_LENGTH = 8 * 1024 * 1024
```

```

ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg', 'gif'])

# OAuth2 configuration.

# This can be generated from the Google Developers Console at
# https://console.developers.google.com/project/_/apiui/credential.

# Note that you will need to add all URLs that your application uses as
# authorized redirect URIs. For example, typically you would add the following:

#
# * http://localhost:8080/oauth2callback
# * https://<your-app-id>.appspot.com/oauth2callback.
#

# If you receive a invalid redirect URI error review you settings to ensure
# that the current URI is allowed.

GOOGLE_OAUTH2_CLIENT_ID = \
    'your-client-id'

GOOGLE_OAUTH2_CLIENT_SECRET = 'your-client-secret'

EOF

# Python environment setup

virtualenv -p python3 /opt/app/7-gce/env

source /opt/app/7-gce/env/bin/activate

/opt/app/7-gce/env/bin/pip install -r /opt/app/7-gce/requirements.txt

```

```
python /opt/app/7-gce/bookshelf/model_cloudsql.py

# Set ownership to newly created account

chown -R pythonapp:pythonapp /opt/app

# Put supervisor configuration in proper place

cp /opt/app/7-gce/python-app.conf /etc/supervisor/conf.d/python-app.conf

# Start service via supervisorctl

supervisorctl reread

supervisorctl update
```

```
output.tf
```

```
output "load_balancer_ip_address" {

    value = module.loadbalancer.loadbalancer_ip
```

```
terraform.tfvars
```

```
project_id = "bookshelf-322313"
```

```
instance_network = "bookshelf-vpc-network"
```

```
region = "europe-west3"
```

```
startup_script = "startup-script.sh"
```

```
db_user = "mainuser"
```

```
gce_instance_type = "e2-medium"
```

```
instance_network_tag = "in-tag"

db_pswd = "Fb9saKDe8rE9gBjb"

db_user_pswd = "Fb9saKDe8rE9gBjb"

sql_region = "europe-west3"

service_account_for_instance = "terraform"

image_bucket_name = "terraform-state-august2021"

bucket_location = "EU"

destroy_policy = "true"
```

```
cd modules/bucket
```

```
main.tf
```

```
resource "google_storage_bucket" "image_storage" {

  name      = var.image_bucket_name

  location  = var.bucket_location

  force_destroy = var.destroy_policy
}


resource "google_storage_bucket_iam_member" "allusers" {

  bucket = google_storage_bucket.image_storage.name

  role = "roles/storage.objectViewer"
```

```
member = "allUsers"
```

```
output.tf
```

```
output "storage_bucket" {
```

```
    value = google_storage_bucket.image_storage
```

```
variable.tf
```

```
variable "image_bucket_name" {}
```

```
variable "bucket_location" {}
```

```
variable "destroy_policy" {
```

```
    type = bool
```

```
}
```

```
cd modules/instance_group
```

```
main.tf
```

```
resource "google_compute_instance_template" "bookshelf_template" {
```

```
    name      = var.template_name
```

```
    description = "Template is used to create bookshelf app server instances"
```

```
tags = [var.instance_tag]
```

```
machine_type = var.instance_machine_type
```

```
can_ip_forward = false
```

```
scheduling {
```

```
    automatic_restart = false
```

```
}
```

```
disk {
```

```
    source_image = var.boot_disc_image
```

```
    auto_delete = true
```

```
    boot = true
```

```
}
```

```
network_interface {
```

```
    network = var.instance_network
```

```
}
```

```
metadata_startup_script = var.startup_script
```

```
service_account {
```

```
    email = var.service_account_for_instance
```

```

    scopes = ["cloud-platform"]

}

}

```

```

resource "google_compute_health_check" "autohealing" {

  name          = "autohealing-health-check"

  check_interval_sec = 5

  timeout_sec      = 5

  healthy_threshold = 2

  unhealthy_threshold = 10

  http_health_check {

    request_path = "/books/"

    port        = "8080"

  }

}

```

```

resource "google_compute_region_instance_group_manager" "bookshelfserver" {

  name = "bookshelfserver-igm"

  base_instance_name = "bookshelf"

```



```
region      = var.igm_region
```

```
version {
```

```
    instance_template = google_compute_instance_template.bookshelf_template.id
```

```
}
```

```
named_port {
```

```
    name = "http"
```

```
    port = "8080"
```

```
}
```

```
# lifecycle {
```

```
#   create_before_destroy = true
```

```
# }
```

```
auto_healing_policies {
```

```
    health_check    = google_compute_health_check.autohealing.id
```

```
    initial_delay_sec = 300
```

```
}
```

```
}
```

```
resource "google_compute_region_autoscaler" "bookshelf_autoscaler" {
```

```
name = "bookshelf-autoscaler"

region = var.igm_region

target = google_compute_region_instance_group_manager.bookshelfserver.id


autoscaling_policy {
  max_replicas = 4
  min_replicas = 2
  cooldown_period = 60

  cpu_utilization {
    target = 1
  }
}

}

output.tf

output "instance_grp" {
  value =
google_compute_region_instance_group_manager.bookshelfserver.instance_group
```

variables.tf

```
variable "instance_network" {}
```

```
variable "startup_script" {}
```

```
variable "service_account_for_instance" {}
```

```
variable "template_name" {}
```

```
variable "instance_machine_type" {}
```

```
variable "igm_region" {}
```

```
variable "instance_tag" {}
```

```
variable "boot_disc_image" {}
```

cd modules/loadbalancer

main.tf

```
resource "google_compute_global_forwarding_rule" "default" {
```

```
  name      = "global-rule"
```

```
target    = google_compute_target_http_proxy.default.id

port_range = "80"

}
```

```
resource "google_compute_target_http_proxy" "default" {

  name      = "target-proxy"

  url_map    = google_compute_url_map.default.id

}
```

```
resource "google_compute_url_map" "default" {

  name          = "url-map-target-proxy"

  default_service = google_compute_backend_service.default.id

}
```

```
resource "google_compute_backend_service" "default" {

  name      = "backend"

  port_name = "http"

  protocol  = "HTTP"

  timeout_sec = 10

  backend {

    group = var.backend_instance_group

  }

}
```

```

    }

    health_checks = [google_compute_http_health_check.default.id]
}

```

```

resource "google_compute_http_health_check" "default" {

    name          = "check-backend"

    request_path   = "/books/"

    check_interval_sec = 1

    timeout_sec    = 1

    port          = 8080
}

```

variables.tf

```
variable "backend_instance_group" {}
```

outputs.tf

```

output "loadbalancer_ip" {

    value = google_compute_global_forwarding_rule.default.ip_address
}

```

cd modules/nat

main.tf

```
resource "google_compute_router" "router" {  
  
  name    = var.router_name  
  
  network = var.selected_network  
  
  region = var.region  
  
}
```

```
resource "google_compute_router_nat" "nat" {  
  
  name                = var.nat_name  
  
  router              = google_compute_router.router.name  
  
  region              = google_compute_router.router.region  
  
  nat_ip_allocate_option = "AUTO_ONLY"  
  
  source_subnetwork_ip_ranges_to_nat =  
  "ALL_SUBNETWORKS_ALL_IP_RANGES"  
  
}
```

variables.tf

```
variable "router_name" {  
  
  default = "default-router"  
  
}
```

```
variable "selected_network" {  
  
}
```

```
variable "nat_name" {  
    default = "default-nat"  
  
}
```

```
variable "region" {  
    default = "europe-west3"
```

```
cd modules/network
```

```
main.tf
```

```
resource "google_compute_network" "bookshelf_vpc_network" {  
    name                = var.vpc_name  
    auto_create_subnetworks = true  
    routing_mode        = "GLOBAL"  
}
```

```
resource "google_compute_global_address" "private_ip_address" {
```

```
  name      = "private-ip-address"
```

```
  purpose   = "VPC_PEERING"
```

```
  address_type = "INTERNAL"
```

```
  prefix_length = 24
```

```
  network    = google_compute_network.bookshelf_vpc_network.id
```

```
}
```

```
resource "google_service_networking_connection" "private_vpc_connection" {
```

```
  network      = google_compute_network.bookshelf_vpc_network.id
```

```
  service      = "servicenetworking.googleapis.com"
```

```
  reserved_peering_ranges =
```

```
[google_compute_global_address.private_ip_address.name]
```

```
}
```

```
resource "google_compute_firewall" "ssh_rule_IAP" {
```

```
  name      = "bookshelf-ssh-firewall-rule"
```

```
  network    = google_compute_network.bookshelf_vpc_network.name
```

```
  source_ranges = ["35.235.240.0/20"]
```



```
target_tags = [var.instance_tag]

allow {

    protocol = "tcp"

    ports    = ["22"]

}

}

resource "google_compute_firewall" "http_rule" {

    name = "bookshelf-http-firewall-rule"

    source_ranges = [

        "130.211.0.0/22",

        "35.191.0.0/16"

    ]

    target_tags = [var.instance_tag]

    network    = google_compute_network.bookshelf_vpc_network.name

    allow {

        protocol = "tcp"

        ports    = ["8080"]

    }

}
```

```
variables.tf
```

```
variable "vpc_name" {}
```

```
variable "instance_tag" {}
```

```
output.tf
```

```
output "bookshelf_vpc" {
```

```
    value = google_compute_network.bookshelf_vpc_network
```

```
}
```

```
cd modules/serv_account
```

```
main.tf
```

```
locals {
```

```
    sa_member = "serviceAccount:$
```

```
{google_service_account.instance_service_account.email}"
```

```
}
```

```
resource "google_service_account" "instance_service_account" {
```

```
    account_id = var.instance_servaccount_id
```

```
}
```

```
resource "google_project_iam_member" "storage_object_admin" {  
  role = "roles/storage.objectAdmin"  
  member = local.sa_member  
}
```

```
resource "google_project_iam_member" "cloudsql_client" {  
  role = "roles/cloudsql.client"  
  member = local.sa_member  
}
```

```
resource "google_project_iam_member" "compute_network_user" {  
  role = "roles/compute.networkUser"  
  member = local.sa_member  
}
```

```
resource "google_project_iam_member" "errorreporting_writer" {  
  role = "roles/logging.logWriter"  
  member = local.sa_member  
}
```

```
resource "google_project_iam_member" "pubsub_editor" {
  role = "roles/pubsub.editor"
  member = local.sa_member
}
```

```
resource "google_project_iam_member" "monitoring_metric_writer" {
  role = "roles/monitoring.metricWriter"
  member = local.sa_member
}
```

```
resource "google_project_iam_member" "source_reader" {
  role = "roles/source.reader"
  member = local.sa_member
}
```

variables.tf

```
variable "instance_servaccount_id" {}
```

output.tf

```
output "custom_instance_service_account" {
```

```
    value = google_service_account.instance_service_account.email
}
```

```
cd modules.sql
```

```
main.tf
```

```
resource "google_sql_database_instance" "bookshelf_db" {

  name          = var.db-instance_name

  database_version = var.db_version

  root_password    = var.db_password

  region          = var.sql_region

  deletion_protection = false

  settings {

    tier = var.machine_type

    ip_configuration {

      ipv4_enabled = false

      private_network = var.selected_network

    }

  }

}
```

```
resource "google_sql_database" "bookshelf_db" {  
  name    = var.bookshelf_database_name  
  instance = google_sql_database_instance.bookshelf_db.id  
}  
  
resource "google_sql_user" "users" {  
  name    = var.db_user_name  
  instance = google_sql_database_instance.bookshelf_db.name  
  password = var.db_user_password  
}  
  
variables.tf  
  
variable "db_version" {}  
  
variable "db-instance_name" {}  
  
variable "db_password" {}  
  
variable "bookshelf_database_name" {}  
  
variable "machine_type" {}
```

```
variable "selected_network" {}
```

```
variable "db_user_name" {}
```

```
variable "db_user_password" {}
```

```
variable "sql_region" {}
```

```
output.tf
```

```
output "connection_name" {
```

```
    value = google_sql_database_instance.bookshelf_db.connection_name
```

```
}
```

ДОДАТОК В ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДОПОВІДІ

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»



Інститут прикладного системного аналізу
Кафедра математичних методів системного
аналізу

Дипломна робота на тему:

**Модель управління ресурсами
гетерогенних баз даних в хмарному
середовищі**

Виконав:

студент IV курсу, групи ІС-зп92, Єрмоленко
С.О.

Науковий керівник:

професор кафедри ММСА,
д.т.н., професор Мухін В.Є.

Київ – 2021

МЕТА ДИПЛОМНОЇ РОБОТИ

Метою даної роботи є дослідження систем управління базами даних, подальшому вибору ефективної моделі управління ресурсами гетерогенних баз даних в хмарному середовищі та здійснення імплементації аплікації з різними видами баз даних та веб-інтерфейсом.

Об'єктом дослідження є процеси створення інфраструктури аплікації з базами даних.

АКТУАЛЬНІСТЬ

Робота з даними, а саме їх зберігання та подальша обробка лежить в основі як економічних моделей світових гігантів електронної комерції так і переважної більшості представників малого та середнього бізнесу. Крім показників фінансової ефективності, компанії в сфері роботи з даними повинні неухильно дотримуватися імплементації положень “Загального регламенту про захист даних” (GDPR – General Data Protection Regulation), що робить роботу з базами даних, в яких ця інформація зберігається, надзвичайно важливим критичним компонентом успішного бізнесу.

ВСТУП

Об’єкт дослідження: процеси створення інфраструктури аплікації з базами даних.

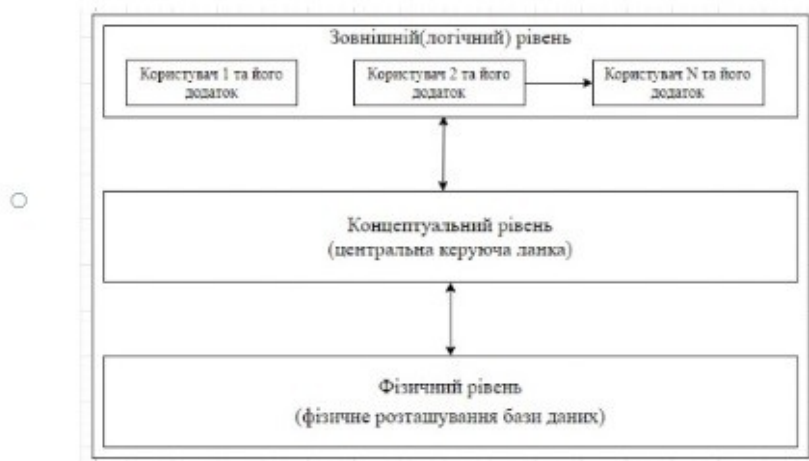
Предмет дослідження: методи автоматизації розгортання інфраструктури аплікації та подальша робота з базами даних.

Методи дослідження: програмні рішення хмарних провайдерів та незалежні інструменти для створення інфраструктури.

ПОСТАНОВКА ЗАДАЧІ

- дослідити існуючі рішення, переваги та обраних хмарних провайдерів;
- дослідити існуючі рішення, переваги та недоліки спеціалізованих інструментів для створення та підтримки інфраструктурних рішень
- спроектувати та імплементувати архітектуру додатка на основі вибору оптимального рішення

МОДЕЛЬ АРХІТЕКТУРИ СУБД



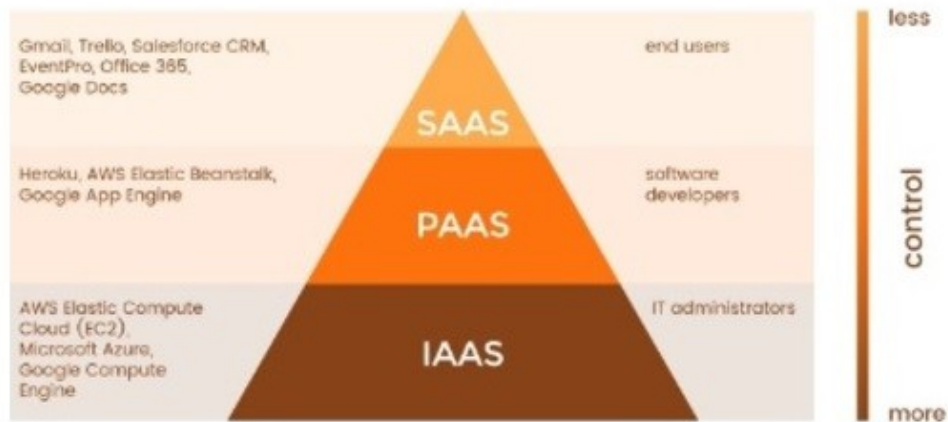
ХАРАКТЕРИСТИКИ МОДЕЛЕЙ УПРАВЛІННЯ РЕСУРСАМИ БД

- трирівнева модель архітектури системи управління базами даних;
- поняття що таке дані, модель даних та моделювання даних;
- поняття системи управління базами даних, типи баз даних;
- програмні комплекси для управління і підтримки ресурсів баз даних в хмарному середовищі;
- сервіси та компоненти для управління та підтримки ресурсів;

РІШЕННЯ ХМАРНИХ ПРОВАЙДЕРІВ

- IaaS — Infrastructure as a Service, рішення інфраструктура як сервіс та його переваги;
- PaaS — Platform as a Service, рішення платформа як сервіс та його переваги;
- SaaS — Software as a Service, рішення програмне забезпечення як сервіс;

СХЕМА ВИКОРИСТАННЯ СЕРВІСІВ



ОГЛЯД ТА АНАЛІЗ СЕРВІСІВ ХМАРНИХ ПРОВАЙДЕРІВ

- Google Cloud Platform, сервіси Cloud Storage, Cloud SQL, Cloud Spanner, Cloud Datastore, Cloud Bigtable;
- Amazon Web Service, сервіси S3, DynamoDB, Redshift, ElastiCache, веб-сервіс RDS який управляє базами даних такими як: Amazon Aurora, MySQL, PostgreSQL, MariaDB;
- Microsoft Azure Platform надає окремі рішення щодо зберігання даних, таких як: Blob Storage, Table Storage, File Storage, Queue Storage, Disk Storage та роботи з базами даних, таких як: SQL Database, SQL Managed Instance, SQL Server on Virtual Machines, Database for PostgreSQL, Database for MySQL, CosmosDB;

ПРОГРАМНІ РІШЕННЯ ВІД ХМАРНИХ ПРОВАЙДЕРІВ

- AWS CloudFormation;
- Azure Resource Manager;
- Google Cloud Deployment Manager;

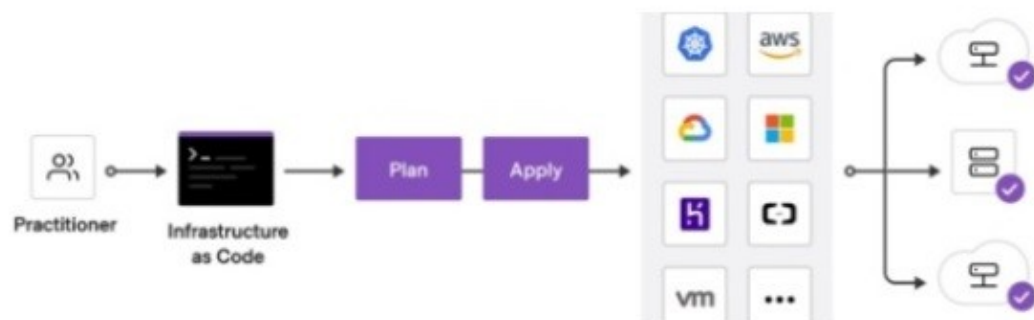
УНІВЕРСАЛЬНІ ІНСТРУМЕНТИ КЕРУВАННЯ ІНФРАСТРУКТУРОЮ

- парадигма IaC — Infrastructure as Code, інфраструктура як код;
- Terraform
- Ansible
- Chef
- Puppet
- Saltstack

ОБРАНА МОДЕЛЬ РЕАЛІЗАЦІЇ ТА ЇЇ ОСОБЛИВОСТІ

- робота з різними хмарними провайдерами;
- декларативний стиль програмування у зручному для розумінні форматі YAML;
- відстеження зміни ресурсів безпосередньо під час розгортання;
- можливість додати власного провайдера;
- Використання багаторазових конструкцій — модулів
- Працює з VCS — version control system, системою контролю версій

СХЕМА РОБОТИ TERRAFORM



ВИСНОВКИ

Метою роботи була розробка моделі управління ресурсами гетерогенних баз даних в хмарному середовищі Google Cloud Platform та підтримка її стабільної роботи з можливостями балансування навантаження та масштабування на випадок різких пікових навантажень. Була спроектована архітектура додатка з функціональним наповненням електронної бібліотеки. В рамках імплементації завдання був ґрунтовно досліджений такий інструмент як Terraform. Шлях до подальшої модернізації додатка лежить в площині автоматизації розгортання шляхом впровадження таких інструментів як Ansible, Chef, Puppet.

ДЯКУЮ ЗА УВАГУ!