Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи моделювання масштабованих хмарних ресурсів

(тема)

Виконав:

студент II курсу, групи СПм-22-6

Поповкін М. М.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Волк М.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Поповкіну Максиму Максимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи моделювання масштабованих хмарних ресурсів

затверджена наказом по університету від ' 01 " квітня 2024 р. № 257 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 червня 2024 р.

3. Вхідні дані до роботи Аккаунт Azure Cloud

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз проблеми та огляд існуючих рішень;
2) вибір технології розробки та інструментальних засобів;
3) розробка алгоритмічного забезпечення;
4) розробка програмних модулів;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд-презентація – 12 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	02.04.24-08.04.24	
2	Вибір технології розробки та	09.04.24-16.04.24	
	інструментальних засобів		
3	Розробка алгоритмічного забезпечення	17.04.24-22.04.24	
4	Розробка програмних модулів	23.04.24-06.05.24	
5	Відлагодження програмних модулів	07.05.24-23.05.24	
6	Оформлення матеріалів кваліфікаційної роботи	24.05.24-03.06.24	
7	Подання кваліфікаційної роботи керівникові та	04.06.24-07.06.24	
	її попередній захист		
8	Подання кваліфікаційної роботи на	08.06.24-12.06.24	
	рецензування		

Дата видачі завдання 01 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Волк М.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 73 с., 7 рис., 1 табл., 1 дод., 13 джерел.

ХМАРНІ ОБЧИСЛЕННЯ, МАСШТАБОВАНІ ХМАРНІ РЕСУРСИ, МЕТОДИ МОДЕЛЮВАННЯ, TERRAFORM, AZURECLOUD, ОПТИМІЗАЦІЯ ІНФРАСТРУКТУРИ, ЕФЕКТИВНЕ ВИКОРИСТАННЯ РЕСУРСІВ, АНАЛІЗ ТА ОЦІНКА, ПРОДУКТИВНІСТЬ, СТАБІЛЬНІСТЬ, ЕФЕКТИВНІСТЬ ВИТРАТ, ІНТЕГРАЦІЯ РЕСУРСІВ, МОДЕЛЮВАННЯ ІНФРАСТРУКТУРИ.

Метою кваліфікаційної роботи є виявлення найбільш ефективних підходів до моделювання хмарних ресурсів, що дозволить організаціям досягти значного зниження витрат на використання хмарних ресурсів при одночасному забезпеченні високого рівня продуктивності та надійності хмарних сервісів.

У ході виконання кваліфікаційної роботи впровадження інноваційних методів оптимізації розподілу ресурсів у масштабованих хмарних середовищах, зокрема на базі Azure Cloud та за допомогою Terraform, дозволило відкрити нові горизонти для підвищення ефективності, надійності та економічності хмарних інфраструктур. Наукова новизна даної роботи полягає у розробці модифікованої методики автоматизації, яка, на відміну від існуючих підходів, включає інтелектуальне масштабування та декларативний опис інфраструктури, що сприяє більш ефективному задоволенню змінних потреб сучасних додатків та сервісів.

ABSTRACT

Master’s thesis pages, 73 figures, 1 tables, 1 appendices, 13 sources.

CLOUD COMPUTING, SCALABLE CLOUD RESOURCES, MODELING METHODS, TERRAFORM, AZURE CLOUD, INFRASTRUCTURE OPTIMIZATION, EFFICIENT USE OF RESOURCES, ANALYSIS AND EVALUATION, PERFORMANCE, STABILITY, COST EFFECTIVENESS, RESOURCE INTEGRATION, INFRASTRUCTURE MODELING.

The major goal of this thesis is to identify the most effective approaches to modeling waste resources, which will allow organizations to achieve a significant reduction in costs for waste resources while simultaneously ensuring high The level of productivity and reliability of dark services.

In order to innovative methods for optimizing the distribution of resources in large-scale environments, based on Azure Cloud and with the help of Terraform, have opened up new horizons for advancement efficiency, reliability and cost-effectiveness of hazardous infrastructures. The scientific novelty of this work lies in the development of a modified automation technique, which, in addition to other approaches, includes intelligent scaling and a declarative description of the infrastructure, which hides more efficiency active satisfaction of the changing needs of daily supplies and services.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 9
1 ОГЛЯД ІСНУЮЧИХ МЕТОДІВ МОДЕЛЮВАННЯ МАСШТАБОВАНИХ ХМАРНИХ РЕСУРСІВ 11
1.1 Методи моделювання хмарних обчисленнь 11
1.1.1 Значення моделювання хмарних ресурсів 11
1.1.2 Підходи до моделювання хмарних ресурсів 12
1.2 Terraform як інструмент моделювання 15
1.2.1 Основні переваги Terraform як інструмента моделювання 16
1.2.2 Компоненти Terraform для моделювання масштабованих хмарних ресурсів 17
1.2.3 Практичні використання Terraform для моделювання 18
1.3 Переваги та обмеження існуючих методів 20
1.3.1 Оцінка переваг та обмежень різних методів моделювання 20
1.3.2 Аналіз популярних інструментів. Критичний розгляд застосування до масштабованих хмарних ресурсів 22
1.4. Напрямки покращення існуючих методів 24
1.4.1 Ідентифікація потенційних покращень 24
1.4.2 Рекомендації та пропозиції щодо оптимізації процесів 26
1.5 Постановка задачі 28
2 ОПИС ТЕХНОЛОГІЙ ТА МЕТОДІВ, ЩО ВИКОРИСТОВУЮТЬСЯ 31
2.1 Azure Api Gateway 31
2.2 Azure Functions 31
2.3 Azure Cosmos DB 32
2.4 Azure Autoscale 33
2.5 Azure Monitor 33
2.6 Infrastructure as Code (IaC) 34

2.7 Методика DevOps 35
3 ПЛАНУВАННЯ ІНФРАСТРУКТУРИ 37
3.1 Інтеграція та автоматизація 38
3.1.1 Впровадження IaC з використанням Terraform 39
3.2 Налаштування та інтеграція компонентів 40
3.2.1 Налаштування Azure API Gateway 40
3.2.2 Налаштування Azure Functions 41
3.2.3 Налаштування Azure SQL Database 41
3.2.4 Інтеграція GitHub Actions 42
3.3 Розробка дизайну системи 42
3.3.1 Основні аспекти дизайну системи 43
3.3.2 Документація та візуалізація 43
3.3.3 Аналіз та стратегія 44
3.3.4 Розробка стратегії масштабування і відновлення 45
4 РОЗРОБКА КОНЦЕПТУАЛЬНОЇ СХЕМИ АРХІТЕКТУРИ 46
4.1 Впровадження IaC з використанням Terraform 47
4.1.1 Організація файлів та директорій 47
4.1.2 Виконання Terraform 49
4.2 Налаштування та інтеграція компонентів 49
4.2.1 Azure API Gateway 49
4.2.2 Розробка функції в Azure Functions 51
4.2.3 Конфігурація баз даних в Azure SQL Database 53
4.2.4 Моніторинг та оптимізація 55
4.2.5 Звітність і сповіщення 56
4.2.6 Тестування та валідація 57
4.3 Впровадження GitHub Actions для CI/CD процесів 59
4.4 Оцінка продуктивності, надійності та економічної ефективності 60
ВИСНОВКИ 63
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 65
ДОДАТОК А Графічний матеріал кваліфікаційної роботи 67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – аплікаційний програмний інтерфейс (англ., Application Programming Interface)

Azure Cloud – хмарна платформа від Microsoft

Blob Storage – бінарне велике об'єктне сховище (англ., Binary Large Object Storage)

CI/CD – неперервна інтеграція та неперервне розгортання (англ., Continuous Integration/Continuous Deployment)

GitHub Actions – платформа CI/CD від GitHub для автоматизації процесів

IaaS – інфраструктура як послуга (англ., Infrastructure as a Service)

IAM – управління ідентифікацією та доступом (англ., Identity and Access Management)

.NET API – програмний інтерфейс, розроблений на платформі Microsoft .NET

Newman – командно-рядковий інструмент для запуску колекцій Postman

Postman – інструмент для тестування API

Service Principal – обліковий запис служби в Azure, який використовується для автоматизованого доступу до ресурсів

SQL – мова структурованих запитів (англ., Structured Query Language)

Terraform – інструмент для збудови, зміни та версіонування інфраструктури безпечно та ефективно

ВСТУП

У сучасному світі хмарні технології стають все більш важливими для управління та розгортання інфраструктури в інформаційних системах. Хмарні обчислення відкривають нові можливості для динамічного аналізу та обробки великих обсягів даних та послуг в Інтернеті. Ці системи забезпечують доступ до розподілених ресурсів, таких як хости, пристрої зберігання даних, процесорні модулі та інші, за запитом користувачів. Використання хмарних обчислень приносить передові технології в обробку та аналіз інформації, що охоплює широкий спектр від робочого столу користувача до великих систем обробки даних. Зазвичай доступ до такої інформації реалізується через веб-додатки на основі хмарної моделі надання послуг.

Хмарні системи представляють собою глобальні мережі датацентрів, що об'єднуються в єдину інфраструктуру високошвидкісними комп'ютерними мережами. Вони можуть ефективно забезпечувати збір, обробку та зберігання інформації. Послуги хмарних систем розподіляються між користувачами з використанням віддалених ресурсів у конкретний момент часу через Інтернет, що забезпечує велику гнучкість та масштабованість для бізнесу та індивідуальних користувачів.

Для ефективного використання ресурсів хмарних систем необхідно вирішити проблему розподілу ресурсів, що полягає у динамічному перерозподілі програмних завдань та потоків даних в реальному часі. Це вимагає від організацій адаптації своїх IT-стратегій для впровадження та використання хмарних рішень максимально ефективно. Розподіл ресурсів у хмарних системах - це процес розподілу віддалених ресурсів через комп'ютерну мережу для виконання відповідних хмарних програм. На високому рівні абстракції такий механізм розподілу отримав назву Інфраструктура як послуга (IaaS), яка надає комп'ютерні ресурси за запитами

користувачів завдяки попередньо визначеному або динамічному механізму розподілу ресурсів.

Цей динамічний механізм розподілу ресурсів дозволяє компаніям та організаціям оптимізувати свої IT-витрати, адаптуючись до змінного навантаження та потреб у реальному часі без необхідності інвестувати в додаткове апаратне забезпечення. Окрім того, хмарні технології сприяють підвищенню рівня безпеки даних, оскільки провайдери хмарних сервісів зазвичай пропонують розширені можливості щодо захисту інформації та відновлення після збоїв.

Впровадження хмарних рішень також відкриває двері до інноваційних підходів у розробці та доставці програмного забезпечення. Завдяки практиці неперервної інтеграції та неперервного розгортання (CI/CD), розробники можуть швидко вносити зміни до своїх додатків і негайно розгортати оновлення, що забезпечує неперевершену швидкість впровадження інновацій та реагування на потреби ринку.

Особливу увагу в сучасному світі хмарних обчислень заслуговує питання стійкості та екологічної відповідальності. Хмарні провайдери, зосереджуючи ресурси в великих дата-центрах та оптимізуючи їх використання, можуть досягати значної економії енергії порівняно з традиційними центрами обробки даних. Це не лише знижує вартість обслуговування IT-інфраструктури для бізнесу, але й сприяє зменшенню вуглецевого сліду, вносячи вклад у боротьбу зі зміною клімату.

Таким чином, хмарні технології представляють собою потужний інструмент для сучасних організацій, що дозволяє їм бути гнучкими, інноваційними та екологічно відповідальними в швидко змінному цифровому світі. Використання хмарних ресурсів відповідно до реальних потреб дозволяє досягти оптимального балансу між продуктивністю, витратами та впливом на довкілля.

1 ОГЛЯД ІСНУЮЧИХ МЕТОДІВ МОДЕЛЮВАННЯ МАСШТАБОВАНИХ ХМАРНИХ РЕСУРСІВ

1.1 Методи моделювання хмарних обчисленнь

Хмарні обчислення, у сучасному IT-середовищі, виступають необхідним та ключовим компонентом для множини організацій, що діють у різноманітних галузях. Забезпечуючи компаніям гнучкість, масштабованість та доступність ресурсів, хмарні платформи стають ідеальним інструментом для вирішення завдань, пов'язаних з обробкою даних, забезпеченням безпеки і підвищенням ефективності використання інфраструктури. Проте, перш ніж розпочати реалізацію масштабованих хмарних ресурсів, важливо провести глибокий аналіз і вибрати оптимальні методи моделювання. Цей етап є вирішальним для ефективного планування та розгортання інфраструктури, а також дозволяє уникнути непередбачених проблем та забезпечити оптимальне використання хмарних ресурсів. У цьому контексті, розділ "Методи моделювання хмарних обчислень" надає можливість ретельного розгляду та вибору належних підходів для досягнення поставленої мети скорочення часу та грошей у процесі проектування масштабованих хмарних ресурсів.

1.1.1 Значення моделювання хмарних ресурсів

Моделювання хмарних ресурсів, у контексті хмарних обчислень, виявляється важливим кроком, оскільки відображає процес вирішення проблем та прийняття стратегічних рішень, пов'язаних із розгортанням та управлінням інфраструктурою. Цей етап є не лише технічною необхідністю, але й стратегічним інструментом, який дозволяє організаціям адаптуватися до швидкозмінних вимог та забезпечує їхню конкурентоспроможність.

У першу чергу, моделювання визначається необхідністю оцінки обсягу робіт та потреб у ресурсах. Розглядаються такі аспекти, як очікувана кількість користувачів, об'єм даних для обробки та інші параметри, які визначають навантаження на інфраструктуру. Це дозволяє точно визначити ресурси, необхідні для задоволення потреб організації.

Далі, моделювання враховує архітектурні вирішення, що визначають оптимальні шляхи розгортання ресурсів та їхню взаємодію. Важливо враховувати, як різні компоненти інфраструктури взаємодіють між собою, як це впливає на продуктивність та доступність.

Моделювання також ставить перед собою завдання передбачення витрат, пов'язаних із розгортанням та управлінням ресурсами. Це дозволяє уникнути фінансових витрат, які можуть виникнути через неправильне оцінювання потреб та необхідності.

Враховуючи ці аспекти, ефективне моделювання стає стратегічним інструментом для розробників та інженерів, які можуть адаптувати інфраструктуру до конкретних потреб та забезпечити її ефективність на різних етапах проекту. Це дозволяє досягти не лише оптимальної продуктивності, але й максимізує користь від використання хмарних ресурсів, забезпечуючи компанії конкурентні переваги на ринку.

1.1.2 Підходи до моделювання хмарних ресурсів

Під час вибору підходів до моделювання хмарних ресурсів, важливо розглянути різноманітні методи, які можуть бути використані для досягнення поставленої мети. Кожен підхід має свої переваги та недоліки, тому вибір конкретної стратегії моделювання повинен враховувати унікальні вимоги та обставини конкретного проекту.

Аналітичне моделювання.

Аналітичне моделювання в контексті хмарних ресурсів ґрунтується на використанні математичних методів та аналізі статистичних даних з метою

раціонального та точного прогнозування продуктивності та взаємодії хмарної інфраструктури. Цей метод дозволяє не лише визначити потрібні ресурси для задоволення потреб проекту, але й враховувати різні сценарії навантаження, що робить його ефективним інструментом для стратегічного планування.

Переваги аналітичного моделювання:

- точність прогнозування, аналітичні методи дозволяють враховувати різноманітні фактори та визначати їх вплив на продуктивність системи, забезпечуючи точне прогнозування вимог до ресурсів;
- гнучкість, аналітичні моделі можуть бути легко адаптовані до різних типів проектів і умов, що робить їх гнучким інструментом для різних викликів;
- глибокий аналіз, аналітичне моделювання дозволяє розглядати систему на рівні деталей, ідеально підходячи для комплексного вивчення різних аспектів хмарної інфраструктури.

Недоліки аналітичного моделювання:

- потрібність точних вихідних даних, для створення точних аналітичних моделей необхідні точні вихідні дані, які можуть бути важко отримати в реальних умовах;
- складність в обробці складних систем, у випадку дуже складних та динамічних систем аналітичне моделювання може стати складним завданням і вимагати значних зусиль;
- обмеження врахування змінюючихся умов, аналітичні моделі можуть бути менш ефективними у врахуванні змін в умовах роботи системи, оскільки вони базуються на попередніх даних та припущеннях.

Аналітичне моделювання залишається потужним інструментом для планування хмарних ресурсів, але вимагає уважного підходу до вибору методів та збору даних для забезпечення його ефективності у конкретному проекті.

Симуляції у контексті моделювання хмарних ресурсів використовують віртуальні моделі для емуляції роботи хмарної інфраструктури. Цей підхід

полягає у створенні віртуального середовища, яке імітує реальні умови та динаміку хмарного сервісу.

Переваги симуляцій:

- можливість вивчення реакції системи на різні умови, симуляції надають можливість вивчити, як хмарна інфраструктура реагує на різні вхідні умови та навантаження без реального впровадження;

- местування прототипів та сценаріїв, за допомогою симуляцій можна ефективно тестувати різні прототипи та варіанти конфігурацій, щоб вибрати найоптимальніший варіант;

- вартість та час, використання симуляцій може значно зекономити час і ресурси порівняно з реальним розгортанням інфраструктури.

Недоліки симуляцій:

- неповна точність, віртуальні моделі можуть не враховувати всі аспекти реальної інфраструктури, що може призвести до неповної точності симуляцій;

- складність створення деталізованих симуляцій, для деталізованих симуляцій, які точно відтворюють реальні умови, може вимагатися значна робота з створення віртуальних моделей;

- обмежена враховуваність змін, симуляції можуть бути менш ефективними, якщо система часто зазнає змін, які важко або неможливо врахувати в симуляційному середовищі.

Симуляції є потужним інструментом для ефективного тестування та оптимізації хмарної інфраструктури. З їх допомогою можна ефективно вивчати реакцію системи на різні умови, проводити тестування та вибирати оптимальні конфігурації перед реальним розгортанням.

Емпіричні методи.

Емпіричні методи моделювання хмарних ресурсів ґрунтуються на зборі та аналізі реальних даних про роботу інфраструктури. Цей підхід використовує фактичні дані з реальних проектів або історичних даних для прогнозування та оптимізації роботи хмарних ресурсів.

Переваги емпіричних методів:

- точність врахування реальних умов, емпіричні методи враховують реальні умови роботи системи, що робить їх ефективними для точного прогнозування та оптимізації;
- найбільш точне відтворення робочого середовища, емпіричні методи найкраще підходять для моделювання систем, які вже існують або використовують історичні дані;
- можливість вивчення реальної продуктивності, засновані на реальних даних, емпіричні методи дозволяють оцінити фактичну продуктивність та здатність інфраструктури.

Недоліки емпіричних методів:

- обмежена прогностичність, емпіричні методи можуть мати обмежену прогностичність, особливо при спробі застосування їх до нових проектів або змінюючихся умов;
- залежність від наявності даних, для ефективного використання емпіричних методів необхідна наявність відповідних та репрезентативних даних, які не завжди доступні;
- неефективність при екстремальних змінах, при раптових та значних змінах умов емпіричні методи можуть не бути достатньо гнучкими для швидкої адаптації.

Емпіричні методи є потужним інструментом для моделювання реальних умов роботи хмарної інфраструктури. З їх допомогою можна вивчати та оптимізувати реальну продуктивність системи, а також прогнозувати її здатність до масштабування та взаємодії зі змінами в умовах роботи.

1.2 Terraform як інструмент моделювання

Terraform, розроблений HashiCorp, є потужним інструментом для автоматизації процесів розгортання та управління інфраструктурою в

хмарних середовищах. Використання Terraform для моделювання масштабованих хмарних ресурсів відкриває широкий спектр можливостей, спрощуючи процес розгортання та забезпечуючи гнучкість у керуванні інфраструктурою.

1.2.1 Основні переваги Terraform як інструмента моделювання

Terraform, як інструмент для моделювання хмарних ресурсів, вирізняється рядом важливих переваг, що роблять його вельми ефективним у процесі проектування та управління інфраструктурою.

Основні переваги включають:

- декларативний підхід, Terraform використовує декларативний підхід до конфігурації, що означає, що ви описуєте бажаний стан інфраструктури, а не послідовність кроків для його досягнення, це спрощує роботу з кодом, полегшуючи розуміння та модифікацію конфігурації;

- масштабованість та гнучкість, Terraform є універсальним інструментом, що підтримує багато провайдерів, включаючи AWS, Azure, Google Cloud, інші хмарні платформи, а також інфраструктурні рішення, такі як VMware чи OpenStack, це дозволяє розгортати та керувати інфраструктурою в різних середовищах, а також масштабувати проекти за потребою;

- модульність, Terraform підтримує використання модулів, що дозволяє групувати та перевикористовувати частини конфігурації, модульність полегшує організацію коду та прискорює розробку, оскільки однакові конфігурації можна використовувати у різних частинах проекту;

- інфраструктура як код (IaC), Terraform сприяє парадигмі Інфраструктури як коду, що дозволяє управляти та версіонувати інфраструктуру так само, як і програмний код, це полегшує відстеження змін, спільну роботу та управління конфігурацією;

- автоматизація та безпека, Terraform надає засоби автоматизації, такі як планування змін та безпечне застосування конфігурації, це дозволяє ефективно та безпечно внести зміни в інфраструктуру, мінімізуючи ризики та помилки;

- активна спільнота та розвиток, Terraform має активну спільноту користувачів і постійно розвивається, наявність багатьох ресурсів, провайдерів та плагінів відкриває широкі можливості для використання та розширення функціоналу.

Terraform є потужним інструментом, який спрощує моделювання та управління хмарною інфраструктурою, забезпечуючи ефективність та гнучкість у процесі розгортання та масштабування проектів.

1.2.2 Компоненти Terraform для моделювання масштабованих хмарних ресурсів

Terraform є інструментом для декларативного опису та управління інфраструктурою "як код".

Основні компоненти Terraform включають:

- провайдери, провайдери – це важливий елемент Terraform, що визначає інфраструктурну платформу (AWS, Azure, Google Cloud тощо), вони надають Terraform інтерфейс для взаємодії з API конкретного провайдера;

- ресурси, ресурси визначають конкретні елементи інфраструктури, які потрібно створити або управляти (VM, бази даних, мережеві компоненти), вони належать певному провайдеру та описуються в конфігураційних файлах;

- модулі, модулі дозволяють групувати та перевикористовувати блоки конфігурації, вони полегшують організацію коду, спрощують використання однакових конфігурацій у різних частинах проекту;

- змінні, змінні використовуються для параметризації конфігурації, вони дозволяють передавати значення між різними частинами конфігурації та між модулями;

- виводи, виводи визначають значення або ресурси, які можна експортувати з конфігурації Terraform, це дозволяє використовувати ці дані в інших проектах чи модулях;

- локальні значення, локальні значення використовуються для обчислення та зберігання проміжних результатів в межах конфігурації, вони допомагають спростити складні вирази та забезпечують більшу зрозумілість коду;

- завдання, завдання використовуються для виконання додаткових дій під час розгортання ресурсів (наприклад, запуск сценаріїв, налаштування додаткових параметрів).

Ці компоненти дозволяють описати потрібну інфраструктуру, а Terraform відповідає за її створення та управління. Користуючись цими компонентами, можна ефективно моделювати та керувати хмарною інфраструктурою.

1.2.3 Практичні використання Terraform для моделювання

Terraform є потужним інструментом для моделювання та управління хмарною інфраструктурою.

Практичні сценарії використання Terraform:

- створення та масштабування груп ресурсів, один із ключових сценаріїв використання Terraform - це створення та масштабування груп ресурсів, таких як серверні ферми чи кластери, Terraform дозволяє легко визначити та налаштувати кількість екземплярів, їхні характеристики та інші параметри;

- управління мережами та з'єднаннями, Terraform дозволяє налаштовувати мережеві компоненти, такі як віртуальні мережі, підмережі,

правила мережевих груп тощо, також можна описувати та керувати з'єднаннями між різними частинами інфраструктури;

- інтеграція з іншими інструментами, Terraform може бути інтегрований з іншими інструментами автоматизації та оркестрації, такими як Ansible, Jenkins, чи Kubernetes, це дозволяє створювати комплексні процеси автоматизації від розгортання до управління інфраструктурою;

- управління конфігураціями, Terraform сприяє парадигмі Інфраструктури як коду (IaC), дозволяючи управляти конфігурацією інфраструктури так само, як і кодом програми, зміни в інфраструктурі вносяться шляхом зміни конфігураційних файлів, а Terraform визначає необхідні зміни та розгортає їх;

- створення та управління резервними копіями, Terraform дозволяє описати та автоматизувати процес створення резервних копій (наприклад, з використанням служби AWS S3 для збереження бекапів), це забезпечує ефективне та автоматизоване управління резервними копіями інфраструктури;

- визначення політик безпеки, Terraform може бути використаний для опису політик безпеки та конфігурації безпеки інфраструктури, визначення правил доступу, налаштування захисту мережі та інші аспекти забезпечення безпеки;

- керування версіями та командною розробкою, Terraform інтегрується з системами керування версіями (наприклад, Git), що дозволяє ефективно вести розробку та відстежувати зміни, різні команди Terraform (init, plan, apply) дозволяють керувати життєвим циклом інфраструктури в командному середовищі.

1.3 Переваги та обмеження існуючих методів

1.3.1 Оцінка переваг та обмежень різних методів моделювання

У данному розділі проводиться аналіз основних методів моделювання масштабованих хмарних ресурсів з метою визначення їх переваг та обмежень. Розглянемо кожен з методів – аналітичне моделювання, симуляції та емпіричні методи – для здійснення об'єктивної оцінки їхньої придатності для проектування та оптимізації масштабованих хмарних інфраструктур.

Переваги аналітичного моделювання:

- точність та прогнозування, аналітичне моделювання дозволяє використовувати математичні методи для точного визначення характеристик системи, забезпечуючи високий рівень точності в оцінці та прогнозуванні масштабованих хмарних ресурсів;

- гнучкість в роботі з даними, здатність використовувати різні математичні підходи дозволяє адаптувати моделі до різноманітних даних та умов;

- детальний аналіз ефективності, аналітичне моделювання дозволяє проводити детальний аналіз ефективності ресурсів та впливу змін на витрати;

- складність моделювання реального середовища, моделі часто враховують лише абстрактні аспекти системи, що може призвести до нечіткості врахування реальних умов та факторів;

- часова та ресурсна інтенсивність, побудова та розвиток складних математичних моделей може бути витратними за ресурсами та часом процесами.

Переваги симуляцій:

- тестування різних сценаріїв, симуляції дозволяють віртуально тестувати та апробувати різні сценарії масштабування хмарних ресурсів;

- можливість вдосконалення архітектури, виявлення проблем в архітектурі перед реальним впровадженням під час розгортання та тестування віртуальних моделей;

- гнучкість у визначенні умов, легкість зміни параметрів та умов для тестування різних сценаріїв без реального впровадження.

Обмеження симуляцій:

- неповнота віртуальних середовищ, неповне врахування деяких аспектів реального хмарного середовища віртуальними симуляціями;

- високі обчислювальні витрати, потреба в значних обчислювальних ресурсах при створенні та використанні великих обсягів симуляцій.

Переваги емпіричних методів:

- базується на реальних даних, використання реальних даних для отримання точних та практичних результатів;

- адаптація до змін в середовищі, здатність адаптуватися до змін в хмарному середовищі та робочому навантаженні;

- надійність та стабільність, більша надійність, оскільки базується на реальних обсягах даних та роботі системи.

Обмеження емпіричних методів:

- залежність від наявності даних, результати емпіричних методів залежать від доступу до реальних даних;

- вимоги до постійного моніторингу, необхідність постійного моніторингу та апдейтів для збереження актуальності результатів; вплив змін в середовищі на результати,

- зміни в хмарному середовищі можуть впливати на результати, тому необхідна постійна корекція методів аналізу даних.

Аналіз різних методів моделювання показав, що кожен з них має свої переваги та обмеження. Аналітичне моделювання надає високу точність, симуляції дозволяють тестувати різні сценарії, а емпіричні методи базуються на реальних даних, забезпечуючи надійність. Вибір методу повинен залежати від конкретних потреб та умов проекту.

1.3.2 Аналіз популярних інструментів. Критичний розгляд застосування до масштабованих хмарних ресурсів

При виборі інструментів для моделювання масштабованих хмарних ресурсів важливо провести критичний аналіз популярних інструментів, який дозволить з'ясувати їхню придатність для вирішення конкретних задач. Нижче наведено більш детальний аналіз основних пунктів цього розділу

Переваги Terraform:

- імперативний та декларативний підхід, Terraform надає можливість використання як імперативного, так і декларативного підходів до конфігурації інфраструктури, що полегшує роботу з ресурсами хмари;
- крос-платформений, підтримка різних хмарних платформ, таких як AWS, Azure, Google Cloud, дозволяє забезпечити універсальність при моделюванні;
- інфраструктура як код, Terraform дозволяє визначати інфраструктуру за допомогою коду, що спрощує автоматизацію та управління конфігурацією.

Обмеження Terraform:

- велика кількість коду для складних конфігурацій, для складних інфраструктурних конфігурацій може знадобитися велика кількість коду, що може ускладнити розробку та обслуговування;
- навчання та поріг входження, вивчення Terraform може вимагати часу та зусиль, особливо для новачків.

Переваги Ansible:

- простота використання, Ansible відомий своєю простотою в налаштуванні та використанні, що робить його привабливим для широкого кола користувачів;
- агент-лесс архітектура, Ansible використовує агент-лесс архітектуру, що полегшує встановлення та налаштування, а також зменшує витрати ресурсів на цільових вузлах.

Обмеження Ansible:

- відсутність вбудованих можливостей для створення та управління хмаровими ресурсами, Ansible не має вбудованих інструментів для моделювання та управління ресурсами хмар;

- залежність від скриптів та модулів, для реалізації певних завдань може знадобитися написання скриптів чи використання додаткових модулів.

Переваги Kubernetes:

- оркестрація контейнерів, Kubernetes забезпечує потужний механізм для розгортання, масштабування та управління контейнеризованими додатками;

- автоматизація процесів, Kubernetes надає автоматизовані засоби для роботи з мережею, моніторингу та масштабуванням, що спрощує управління ресурсами.

Обмеження Kubernetes:

- складність налаштування та розгортання, розгортання та налаштування Kubernetes може бути складним завданням, особливо для початківців;

- великий обсяг конфігураційного коду, для складних конфігурацій великий обсяг YAML-коду може призвести до ускладнення розробки та обслуговування.

Переваги Azure Resource Manager (ARM) Templates:

- інтеграція з Azure, ARM Templates є нативним інструментом для роботи з ресурсами в середовищі Azure, забезпечуючи глибоку інтеграцію та підтримку хмарної платформи;

- декларативний підхід, ARM використовує декларативний підхід до опису інфраструктури, що полегшує визначення та управління ресурсами.

Обмеження Azure Resource Manager (ARM) Templates:

- залежність від екосистеми Azure, використання ARM Templates виправдане при роботі в хмарному середовищі Azure, але може стати складнішим для мультихмарних або гібридних рішень;

- великий обсяг коду для складних конфігурацій, для складних інфраструктурних конфігурацій може знадобитися велика кількість коду, що ускладнює розробку та обслуговування.

Переваги AWS CloudFormation:

- інтеграція з AWS, CloudFormation є інструментом, нативно підтримуваним AWS, що забезпечує зручну інтеграцію та управління ресурсами хмарної платформи Amazon;

- декларативний підхід, CloudFormation використовує декларативний підхід, що полегшує визначення та управління ресурсами AWS.

Обмеження AWS CloudFormation:

- залежність від екосистеми AWS, використання CloudFormation логічно для проектів, які повністю опираються на хмарні ресурси AWS, але може стати складнішим у гібридних чи мультихмарних середовищах;

- специфічність мови шаблонів, шаблони CloudFormation вимагають специфічного синтаксису AWS, що може виявитися менш універсальним для інших хмарних платформ.

Аналіз інструментів моделювання, таких як Terraform, Ansible, Kubernetes, ARM Templates та CloudFormation, дозволяє зрозуміти, що кожен з них має свої переваги та обмеження.

1.4. Напрямки покращення існуючих методів

1.4.1 Ідентифікація потенційних покращень

У данному розділі проводиться ідентифікація потенційних покращень існуючих методів моделювання масштабованих хмарних ресурсів з метою визначення їх переваг та обмежень. Розглянемо кожен з напрямків.

Оптимізація конфігураційного коду:

- використання модульності, проблема – збільшений обсяг коду, розробка та управління великим обсягом коду у конфігураційних файлах може бути ускладненою та заплутаною;

- можливі покращення - модульний підхід та повторне використання коду, розділення конфігураційного коду на логічні модулі, кожен з яких відповідає за певні аспекти інфраструктури, що дозволяє зручно використовувати та перевикористовувати частини коду для різних проектів чи складних сценаріїв;

- використання шаблонів, проблема – зайві повторення коду, повторення однотипних або схожих конфігурацій може вести до великого обсягу надлишкового коду;

- можливі покращення – створення шаблонів та параметризація, визначення загальних шаблонів для часто використовуваних конфігурацій, що спрощує створення нових ресурсів та скорочує обсяг коду, використання параметрів у шаблонах для адаптації конфігурації до конкретних потреб;

- використання інструментів автоматизації, проблема – велика складність конфігурації, необхідність вручну налаштовувати багато деталей конфігурації може зробити процес управління інфраструктурою більш складним;

- можливі покращення – використання інструментів автоматизації, застосування інструментів, таких як Terraform, які надають можливості автоматизації конфігураційного коду та управління інфраструктурою;

- автоматизація масштабування, проблема – складність ручного масштабування, ручне збільшення або зменшення кількості ресурсів може бути недоцільним та вимагати значних зусиль;

- можливі покращення – використання інструментів автоматизації, застосування інструментів, таких як Kubernetes або Autoscaling в хмарних платформах, для автоматичного реагування на зміни навантаження, сценарії масштабування, розробка чітких сценаріїв автоматичного масштабування для різних умов та ситуацій;

- гнучка стратегія масштабування, проблема – відсутність гнучкості в стратегії масштабування, наявні стратегії можуть бути неефективними у різних умовах та варіантах навантаження;

- можливі покращення – адаптивні стратегії, визначення гнучких стратегій масштабування, які можуть адаптуватися до різних умов, враховуючи, наприклад, часові рамки та природу завдань;

- ефективна робота з контейнерами, проблема – незручності управління контейнерами, управління контейнерами може бути заплутаним та вимагати значних ресурсів;

- можливі покращення – оптимізація оркестраторів контейнерів, використання оркестраторів, таких як Kubernetes, та вдосконалення їх конфігурації для оптимального управління контейнерами;

- ефективне використання ресурсів, проблема – надмірне або недостатнє виділення ресурсів, неоптимальне використання ресурсів може призводити до додаткових витрат або неефективності системи;

- можливі покращення – моніторинг та оптимізація, впровадження систем моніторингу та оптимізації для постійного аналізу та покращення ефективності використання ресурсів;

- автоматична оптимізація, розробка механізмів автоматичної оптимізації, які реагують на зміни у навантаженні та ресурсному споживанні.

Оптимізація конфігураційного коду включає в себе використання модульності, шаблонів та інструментів автоматизації для полегшення розробки та підтримки інфраструктури. Ці покращення сприяють кращій читабельності, повторному використанню коду та зменшенню його обсягу, що робить процес моделювання більш ефективним та керованим.

1.4.2 Рекомендації та пропозиції щодо оптимізації процесів

У даному підрозділі розглядаються ключові аспекти покращення існуючих методів моделювання та управління масштабованими хмарними

ресурсами. Зосереджуючись на конкретних виправленнях та оптимізаціях, цей розділ пропонує рекомендації та пропозиції для оптимізації процесів створення та управління хмарною інфраструктурою.

Аналізуючи існуючі методи моделювання та визначаючи їхні обмеження, цей розділ ставить перед собою завдання пропонувати практичні покращення, спрямовані на забезпечення ефективності, безпеки та оптимальної роботи масштабованих хмарних ресурсів. Розглядаються рекомендації для оптимізації конфігураційного коду, покращення процесів масштабування та ефективного використання ресурсів для досягнення високої продуктивності та вартості операцій в хмарному середовищі.

Можливі покращення:

- стандартизація коду, впровадження стандартизованих правил написання конфігураційного коду для забезпечення читабельності та однорідності;

- використання автоматизованих інструментів для перевірки коду на відповідність стандартам;

- перевірка на помилки та оптимізація, регулярна перевірка конфігураційного коду на наявність помилок та потенційних проблем;

- визначення та усунення дублювання коду для забезпечення ефективного використання ресурсів;

- використання шаблонів та модулів, впровадження шаблонів та модулів для стандартизації та полегшення процесу написання конфігураційного коду;

- розробка бібліотеки з повторно використовуваними елементами для швидкого створення нових конфігурацій;

- динамічне масштабування, використання систем, що автоматично адаптують кількість ресурсів в залежності від навантаження;

- налаштування алгоритмів для швидкого реагування на зміни навантаження та оптимального використання ресурсів;

- сценарії масштабування, розробка детальних стратегій масштабування для різноманітних сценаріїв навантаження;

- визначення порогових значень, за якими відбувається автоматичне масштабування, та забезпечення стійкості системи під час змін;

- моніторинг та аналіз ефективності, встановлення систем моніторингу для стеження за ефективністю масштабування та виявлення можливих проблем;

- використання аналітичних інструментів для оцінки впливу масштабування на продуктивність системи;

- оптимізація розподілу ресурсів, аналіз та оптимізація розподілу ресурсів між різними компонентами інфраструктури;

- застосування алгоритмів, що автоматично визначають оптимальний розподіл ресурсів в залежності від навантаження;

- автоматичне відключення неактивних ресурсів, розробка механізмів, які автоматично відключають неактивні ресурси для зменшення витрат;

- визначення правил для автоматичного відновлення ресурсів при збільшенні навантаження;

- стратегії енергозбереження, використання стратегій енергозбереження для зменшення споживання електроенергії;

- розробка та впровадження методів для автоматичного переходу у режими зменшеного споживання ресурсів у періоди неактивності.

Ці рекомендації спрямовані на покращення існуючих методів оптимізації конфігураційного коду, процесів масштабування та ефективного використання ресурсів для забезпечення оптимальної та стійкої масштабованості хмарних ресурсів.

1.5 Постановка задачі

Метою даного проекту є виявлення найбільш ефективних підходів до моделювання хмарних ресурсів, що дозволить організаціям досягти значного

зниження витрат на використання хмарних ресурсів при одночасному забезпеченні високого рівня продуктивності та надійності хмарних сервісів. В роботі пропонуються ефективні стратегії автоматизації розгортання та управління хмарною інфраструктурою на основі платформи Azure Cloud та інструменту Terraform. Сучасні дослідження, підкреслюють важливість інтеграції автоматизованих інструментів управління для підвищення ефективності використання хмарних ресурсів, що включає аналіз поточних викликів у масштабуваних хмарних ресурсах, таких як балансування навантаження, забезпечення неперервної доступності сервісів та оптимізація використання ресурсів. Огляд методів оптимізації демонструє стратегії зниження витрат та покращення продуктивності в хмарних середовищах. Результати дослідження призначені для широкого кола фахівців у галузі.

Для досягнення поставленої мети перед проектом стоять наступні задачі:

- дослідження поточного стану хмарної інфраструктури, аналіз існуючих хмарних ресурсів, використання та витрат, для виявлення потенціалу оптимізації;
- проектування оптимізованої хмарної архітектури, розробка архітектури на основі Azure Cloud, що включає в себе автомасштабування, балансування навантаження та інші методи для оптимізації використання ресурсів;
- впровадження Infrastructure as Code (IaC), розгортання та управління інфраструктурою через Terraform для забезпечення швидкого, безпечного та відтворюваного процесу розгортання;
- тестування та валідація інфраструктури, перевірка ефективності розробленої інфраструктури через комплексні тести, що імітують реальні сценарії використання, для оцінки продуктивності, надійності та економічної ефективності;

- документування та передача знань, підготовка документації та керівництв по використанню оптимізованої хмарної інфраструктури для забезпечення легкості управління та підтримки.

Проект передбачає використання таких компонентів Azure Cloud, як Azure Api Gateway для маршрутизації запитів, Azure Functions для виконання серверлес обчислень, Azure Cosmos DB або Azure SQL Database для зберігання даних, а також інтеграцію з GitHub Actions для автоматизації CI/CD процесів. Ключову роль відіграватимуть також Azure Monitor та Azure Security Center для моніторингу та забезпечення безпеки хмарної інфраструктури.

2 ОПИС ТЕХНОЛОГІЙ ТА МЕТОДІВ, ЩО ВИКОРИСТОВУЮТЬСЯ

2.1 Azure Api Gateway

Azure API Gateway є частиною облачної платформи Microsoft Azure, яка функціонує як ворота між зовнішнім світом і вашими облачними сервісами, дозволяючи ефективно управляти, маршрутизувати та захищати запити до ваших веб-сервісів і API. Цей сервіс допомагає впроваджувати різноманітні сценарії використання, зокрема, агрегацію декількох сервісів у єдиний API, версіонування API, моніторинг та аналітику, а також контроль доступу до API.

Основні функції:

- маршрутизація запитів, Azure API Gateway дозволяє налаштовувати маршрути для запитів до різних API або мікросервісів на основі URL-адрес, HTTP-методів та інших параметрів запитів;
- управління версіями API, сервіс підтримує версіонування API, дозволяючи одночасно підтримувати декілька версій вашого API та плавно переходити між ними;
- автентифікація та авторизація, інтеграція з Azure Active Directory та іншими провайдерами ідентифікації дозволяє налаштувати автентифікацію та авторизацію для користувачів і додатків, що доступають до API;
- обмеження та квоти, API Gateway надає можливість встановлення обмежень на частоту викликів API для захисту від перевантаження і забезпечення рівномірного розподілу ресурсів.

2.2 Azure Functions

Azure Functions – це обчислювальний сервіс, що дозволяє виконувати код у відповідь на події без необхідності займатися управлінням серверною

інфраструктурою, що робить його ідеальним рішенням для розробки серверлес-додатків. Azure Functions підтримує широкий спектр мов програмування, включаючи C#, Java, JavaScript, TypeScript, Python і PowerShell.

Основні характеристики:

- подієво-орієнтоване виконання, виклик коду можливий за допомогою різних подій, включаючи HTTP-запити, повідомлення з черги та зміни в базі даних;

- широкий вибір тригерів і прив'язок, можливість інтеграції з багатьма Azure сервісами і зовнішніми джерелами за допомогою налаштовуваних тригерів і прив'язок;

- масштабування за потребою, автоматичне масштабування для обробки зростаючого або зменшуваного навантаження без необхідності вручну управляти інфраструктурою;

- підтримка багатьох мов програмування, підтримка C#, Java, JavaScript, TypeScript, Python, і PowerShell, дозволяючи розробникам використовувати знайомі мови.

2.3 Azure Cosmos DB

Azure Cosmos DB – це глобально розподілена база даних як сервіс, що підтримує множинні моделі даних.

Основні характеристики Azure Cosmos DB:

- глобальне розподіл, легке створення глобально розподілених і доступних застосунків з автоматичним масштабуванням пропускної здатності;

- гнучкість моделей даних, підтримка документів, ключ-значення, графів, і сімейств стовпців з множинними API для кожної моделі;

- гарантії консистентності на вибір, п'ять рівнів консистентності, від сильної до затриманої, для оптимального балансу між швидкістю читання і точністю даних;

- автоматичне індексування, всі дані автоматично індексуються, що дозволяє швидко виконувати запити без потреби в ручному управлінні індексами.

2.4 Azure Autoscale

Azure Autoscale – це функціонал, який дозволяє автоматично масштабувати кількість віртуальних машин або інших ресурсів в Azure сервісах на основі актуального навантаження або графіків.

Основні функції:

- автоматичне масштабування, дозволяє додавати або видаляти ресурси для відповідності поточному навантаженню, оптимізуючи витрати та продуктивність;

- налаштування правил масштабування, можливість визначення специфічних метрик, таких як ЦП або пам'ять, які використовуються для масштабування ресурсів;

- графіки масштабування, встановлення графіків для автоматичного масштабування, наприклад, під час пікових годин або сезонних змін у навантаженні.

2.5 Azure Monitor

Azure Monitor надає всебічні можливості моніторингу для забезпечення видимості про стан застосунків, інфраструктури та мережі у вашому Azure середовищі.

Основні функції:

- збір та аналіз даних, збір метрик та журналів з різних джерел, включаючи віртуальні машини, бази даних та застосунки;
- сповіщення, налаштування сповіщень на основі певних метрик або подій для раннього виявлення та вирішення проблем;
- візуалізація, створення наочних панелей для відображення ключових показників продуктивності та стану систем.

2.6 Infrastructure as Code (IaC)

Infrastructure as Code (IaC) – це підхід до управління та автоматизації інфраструктури через код, що дозволяє розробникам та системним адміністраторам використовувати код для налаштування, розгортання та управління інфраструктурою в автоматизований і відтворюваний спосіб. Terraform від HashiCorp – це відкритий інструмент, що дозволяє легко та ефективно застосовувати IaC, підтримуючи численні облачні платформи, включаючи Azure, AWS, Google Cloud Platform та інші.

Основні функції:

- декларативний код для опису інфраструктури, замість того, щоб використовувати набори команд або інтерфейси для налаштування ресурсів, Terraform дозволяє визначати інфраструктуру за допомогою простої та інтуїтивно зрозумілої мови опису (HCL – HashiCorp Configuration Language);
- ідемпотентність, Terraform гарантує, що запуск коду інфраструктури кілька разів поспіль не змінить стан ресурсів, якщо не було змін у конфігурації, це забезпечує стабільність і передбачуваність при розгортанні інфраструктури;
- управління залежностями між ресурсами, Terraform автоматично визначає залежності між ресурсами та розгортає їх у відповідному порядку для забезпечення правильної конфігурації інфраструктури;

- планування та підтвердження змін, перед застосуванням змін Terraform надає можливість переглянути "план" змін, які будуть впроваджені, дозволяючи користувачам підтвердити дії перед їх виконанням;

- модульність і повторне використання коду, Terraform сприяє використанню модулів – пакетів конфігурації, які можна використовувати повторно в різних проектах, що значно спрощує управління складною інфраструктурою.

Використання Terraform для IaC дозволяє значно знизити час та зусилля, необхідні для розгортання і управління інфраструктурою, забезпечуючи високий рівень автоматизації, ефективності та мінімізуючи ризик людських помилок.

2.7 Методика DevOps

Методика DevOps об'єднує розробників програмного забезпечення (Dev) та IT-операції (Ops) для покращення співпраці та продуктивності шляхом автоматизації інфраструктури, робочих процесів та неперервного вимірювання продуктивності розробки програмного забезпечення. Основні елементи методики DevOps включають:

- CI/CD пайплайни за допомогою Azure DevOps, автоматизація процесів неперервної інтеграції (CI) та неперервного розгортання (CD) для швидкого внесення змін та оновлень сервісів, Azure DevOps надає інструменти для створення та керування пайплайнами, які автоматично збирають, тестують і розгортають код у різних середовищах;

- контейнеризація за допомогою Docker, упаковка додатків і залежностей у контейнери для легкості розгортання та масштабування, Docker дозволяє розробникам створювати легковісні, переносні контейнери, які містять усе необхідне для запуску програми, забезпечуючи консистентність середовища від розробки до продакшну;

- оркестрація контейнерів за допомогою Kubernetes, автоматизація розгортання, масштабування та управління контейнеризованими додатками, Kubernetes є відкритим інструментом для оркестрації контейнерів, який дозволяє автоматично розподіляти навантаження, керувати здоров'ям додатків та забезпечувати їх безперервну доступність.

Ці практики DevOps допомагають забезпечити швидку та ефективну розробку та розгортання програмного забезпечення, зменшуючи ризик помилок, покращуючи якість продукції та збільшуючи задоволеність клієнтів.

3 ПЛАНУВАННЯ ІНФРАСТРУКТУРИ

На етапі планування інфраструктури було розроблено концептуальну архітектуру хмарної інфраструктури, яка включала визначення ключових компонентів для оптимальної роботи системи. Основні елементи архітектури обрано згідно з потребами проекту та можливостями Azure Cloud.

На етапі планування інфраструктури розроблено концептуальну архітектуру хмарної інфраструктури, зосереджуючи увагу на інтеграції ключових компонентів, що забезпечують не тільки функціональність, але й високу надійність та безпеку системи. Основна мета цього етапу — створити міцну основу, яка дозволить ефективно управляти запитами та даними в масштабованому середовищі.

Розробка архітектури включала визначення основних технологій:

- azure API Gateway був обраний як центральний елемент управління API через його можливість ефективно маршрутизувати запити, забезпечуючи при цьому високий рівень безпеки, використання даного компонента дозволяє централізовано управляти всіма запитами, що надходять та виходять з хмарних сервісів, а також легко інтегрувати нові сервіси в існуючу структуру;
- azure Functions було обрано для реалізації серверлес обчислень, що значно спрощує розгортання коду без необхідності керування серверною інфраструктурою, це рішення ідеально підходить для обробки подій в реальному часі та масштабується відповідно до потреб системи;
- azure SQL Database забезпечує стабільне та ефективне зберігання даних, з високою продуктивністю та відмовостійкістю, що є критично важливим для будь-якої бізнес-системи.

3.1 Інтеграція та автоматизація

Ефективна інтеграція та автоматизація інфраструктурних компонентів є ключовими для забезпечення швидкості та ефективності розгортання, а також для підтримки безперервного циклу інновацій і вдосконалень. Цей етап охоплює впровадження інфраструктури як коду (IaC) з використанням Terraform, а також детальну настройку кожного компонента системи для забезпечення їх оптимальної роботи та взаємодії.

Впровадження Terraform як інструменту IaC дозволило стандартизувати та автоматизувати розгортання інфраструктури, знижуючи ризик людських помилок та забезпечуючи повторюваність та документування інфраструктурних рішень. Такий підхід забезпечив:

- можливість швидко і точно реплікувати налаштування середовищ забезпечила зменшення часу на розгортання та збільшення продуктивності розробки;
- кожна зміна в інфраструктурі контролюється через систему контролю версій, що дозволяє відслідковувати впровадження змін та швидко відновлювати попередні стани;
- автоматизація процесів оновлення і масштабування ресурсів сприяла більш гнучкому та відповідальному управлінню інфраструктурою.

Для кожного компонента системи було розроблено детальні конфігураційні плани, що забезпечують їх оптимальну роботу та інтеграцію з іншими частинами інфраструктури:

- включаючи внутрішнє та зовнішнє балансування навантаження, налаштування приватних і публічних мережевих зон для забезпечення безпеки та доступності сервісів;
- впровадження політик безпеки, шифрування даних та налаштування систем ідентифікації і авторизації для забезпечення захисту інформації та систем;

- аналіз і налаштування використання обчислювальних, зберігаючих і мережевих ресурсів для оптимальної продуктивності та зниження витрат.

Ці заходи забезпечили не тільки технічну ефективність системи, але й відповідали вимогам бізнесу щодо швидкості внесення змін, масштабування та реагування на нові бізнес-виклики, сприяючи неперервному циклу інновацій і вдосконалення.

3.1.1 Впровадження IaC з використанням Terraform

Terraform було обрано як основний інструмент для IaC завдяки його здатності ефективно керувати ресурсами Azure. Terraform дозволяє описати необхідну інфраструктуру в декларативному стилі, що значно спрощує процес розгортання та управління інфраструктурою.

Створено конфігураційні файли для кожного з компонентів. Наприклад, для Azure API Gateway налаштовані правила маршрутизації, аутентифікації та доступу, які визначаються у відповідних блоках Terraform.

Лістинг 3.1 – Конфігурація Terraform:

```
resource "azurerm_api_management" "example" {
  name                = "example-apim"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  publisher_name      = "example-publisher"
  publisher_email     = "publisher@example.com"

  sku_name = "Developer_1"
}
```

Виконання змін через Terraform, terraform plan використовується для перевірки змін перед застосуванням, а terraform apply для розгортання змін у хмарному середовищі. Ці команди забезпечують прозорість змін і знижують ризик помилок під час розгортання.

3.2 Налаштування та інтеграція компонентів

Після впровадження інфраструктури через Terraform, наступний важливий етап полягає у налагодженні інтеграції між компонентами та їхнім налаштуванні для забезпечення оптимальної роботи системи. Цей процес включає детальне конфігурування кожного сервісу, а також перевірку їхньої взаємодії для досягнення високої ефективності та безпеки.

3.2.1 Налаштування Azure API Gateway

Azure API Gateway використовується для управління API, забезпечення безпеки та маршрутизації запитів. Ключові аспекти налаштування, які були впроваджені, включають:

- політики безпеки, було налаштовано автентифікацію та авторизацію, використовуючи такі механізми, як OAuth2 та Azure Active Directory, це забезпечило контроль доступу до API, дозволяючи лише авторизованим користувачам та системам використовувати функціонал API;
- маршрутизація та версіонування API, були встановлені правила для розподілу трафіку між різними версіями API, що дозволило гнучко управляти розгортаннями без перерви в обслуговуванні, це гарантувало, що зміни можна було впроваджувати поступово, без впливу на існуючих користувачів;
- налаштування збору метрик і логів було здійснено для аналізу використання API та виявлення можливих проблем, це включало трекінг запитів, відповідей та помилок, що дозволяло оперативно реагувати на інциденти та оптимізувати роботу системи;
- для запобігання зловживанням та забезпечення рівномірного навантаження на систему були введені обмеження на кількість запитів від одного користувача за певний проміжок часу;

- Azure API Gateway було налаштовано для спілкування з іншими сервісами, такими як Azure Functions і Azure Logic Apps, забезпечуючи плавну інтеграцію та взаємодію компонентів системи;
- всі дані, що передаються через API Gateway, було автоматично шифровано для забезпечення їх конфіденційності та безпеки;
- ці заходи забезпечили не тільки технічну ефективність системи, але й відповідали вимогам бізнесу щодо безпеки, стабільності та гнучкості в управлінні API.

3.2.2 Налаштування Azure Functions

Azure Functions дозволяє виконувати код у відповідь на різноманітні події, що забезпечує велику гнучкість у розробці безсерверних додатків. Налаштування включає:

- тригери та прив'язки, конфігурація тригерів, таких як HTTP, таймери або події з черг, що ініціюють виконання функцій, прив'язки дозволяють легко взаємодіяти з іншими сервісами, наприклад, базами даних або системами зберігання;
- масштабування, налаштування автоматичного масштабування для реагування на коливання навантаження, забезпечуючи високу доступність і оптимізацію витрат;
- управління залежностями, конфігурація середовища виконання, установка необхідних бібліотек та пакетів.

3.2.3 Налаштування Azure SQL Database

Azure SQL Database — це масштабована, надійна база даних з повним спектром можливостей управління реляційними даними. Ключові елементи налаштування включають:

- безпека даних, налаштування шифрування даних у спокої та під час передачі, налаштування політик доступу для забезпечення захисту даних;
- резервне копіювання та відновлення, конфігурація регулярного резервного копіювання і стратегій відновлення для забезпечення стійкості даних до втрати або пошкодження;
- моніторинг та налаштування продуктивності, використання вбудованих інструментів Azure для моніторингу продуктивності бази даних і оптимізації запитів.

3.2.4 Інтеграція GitHub Actions

GitHub Actions використовується для автоматизації пайплайнів CI/CD, що дозволяє швидко та безпечно розгортати оновлення. Налаштування включає:

- автоматизація робочих процесів, створення скриптів для автоматичного тестування, збірки та розгортання додатків;
- інтеграція з Terraform, використання Actions для запуску Terraform скриптів, що дозволяє автоматизувати процес розгортання інфраструктури.

Ці етапи налаштування та інтеграції забезпечують створення надійної, безпечної та легко масштабованої хмарної інфраструктури, оптимізуючи процеси управління та впровадження.

3.3 Розробка дизайну системи

Дизайн системи був ретельно розроблений для забезпечення ефективної взаємодії між усіма компонентами хмарної інфраструктури. Цей процес включав створення детальних схем, які ілюструють взаємозв'язки та залежності між різними сервісами, а також визначення методів для їх оптимальної інтеграції та масштабування.

3.3.1 Основні аспекти дизайну системи

Логічна структура. Логічна структура системи була розроблена для забезпечення чіткого розуміння функціональних блоків та їх взаємодії. Це включало визначення ролей кожного компоненту в системі, таких як Azure API Gateway для маршрутизації запитів, Azure Functions для обробки подій, та Azure SQL Database для управління даними.

Фізична структура. Фізична структура була спроектована для забезпечення високої доступності та масштабування системи. Було враховано географічне розміщення серверів, а також стратегії реплікації даних між регіонами для зниження латентності та підвищення стійкості системи.

Інтеграція компонентів. Важливим аспектом була інтеграція всіх компонентів системи за допомогою визначених інтерфейсів і протоколів. Наприклад, Azure Functions були інтегровані з Azure API Gateway через HTTP тригери, а бази даних були підключені через засоби безпечного доступу.

Стратегії масштабування та відновлення. Були розроблені стратегії для автоматичного масштабування компонентів у відповідь на змінені навантаження та відновлення системи після потенційних збоїв. Це забезпечує, що система може ефективно впоратися з піковими навантаженнями та швидко відновлюватися без значних перерв у роботі.

3.3.2 Документація та візуалізація

Для наочного представлення архітектури системи були створені діаграми, які ілюструють основні компоненти та їх взаємозв'язки. Ці діаграми служать засобом для забезпечення зрозумілості структури системи для всіх учасників проекту та допомагають у визначенні потреб у модифікації або розширенні системи.

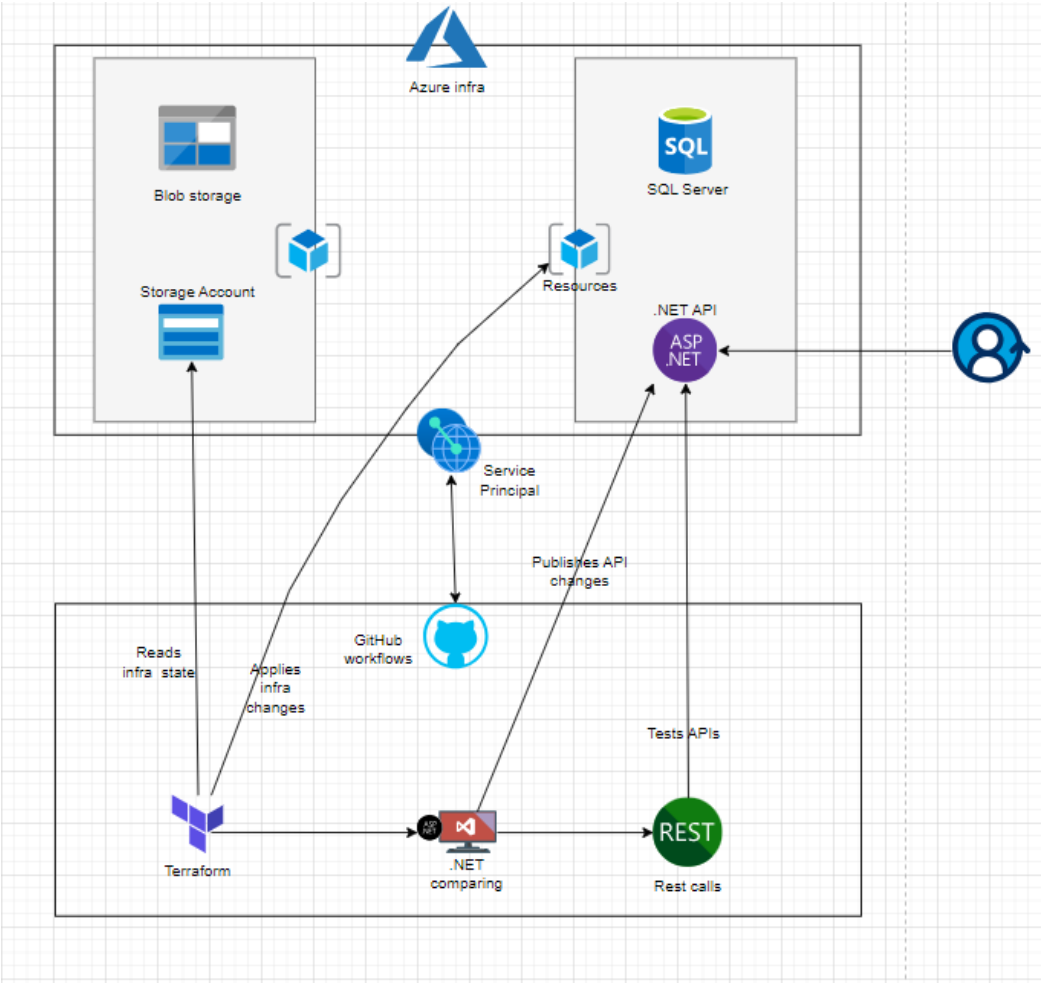

Рисунок 3.1 – Логічна структура системи

Цей етап розробки дизайну системи забезпечив створення міцної основи для подальшого ефективного впровадження та оптимізації хмарної інфраструктури, забезпечуючи необхідну гнучкість для адаптації до мінливих вимог бізнесу.

3.3.3 Аналіз та стратегія

Після розробки дизайну системи було важливо провести глибокий аналіз і стратегічне планування, щоб забезпечити, що архітектура відповідає довгостроковим цілям організації та може адаптуватися до майбутніх технологічних тенденцій та змін у бізнес-вимогах.

Аналіз поточних і майбутніх вимог:

- поточні потреби, було проаналізовано поточні операційні вимоги системи, зокрема необхідність високої доступності, швидкості відгуку, безпеки даних та здатності до масштабування, виявлення цих потреб допомогло забезпечити, що система зможе ефективно обробляти поточний обсяг даних та запитів;

- майбутні тенденції та технології, проаналізовано майбутні технологічні тенденції, які можуть вплинути на систему, включаючи нові можливості хмарних обчислень, штучний інтелект, машинне навчання та великі дані, це забезпечило підготовку до інтеграції нових функцій та можливостей у майбутньому.

3.3.4 Розробка стратегії масштабування і відновлення

Політики безпеки були розроблені для захисту даних від несанкціонованого доступу та атак. Використання шифрування, автентифікації та контролю доступу в Azure Security Center допомагає забезпечити конфіденційність та цілісність даних.

Моніторинг і реагування на інциденти. Azure Monitor використовується для постійного моніторингу діяльності системи, що дозволяє швидко виявляти та реагувати на потенційні загрози безпеці.

Ці стратегічні кроки не тільки підвищують ефективність і надійність існуючої системи, але й забезпечують її готовність до адаптації у відповідь на майбутні зміни та виклики, зберігаючи високий рівень сервісу та відповідність сучасним вимогам до хмарних інфраструктур.

4 РОЗРОБКА КОНЦЕПТУАЛЬНОЇ СХЕМИ АРХІТЕКТУРИ

Розробка концептуальної схеми архітектури була критичним етапом проекту, який забезпечив чітке бачення структури інфраструктури та взаємодії між компонентами. Схема архітектури, яка була розроблена, відображає комплексний підхід до організації ресурсів та послуг у віртуальній мережі.

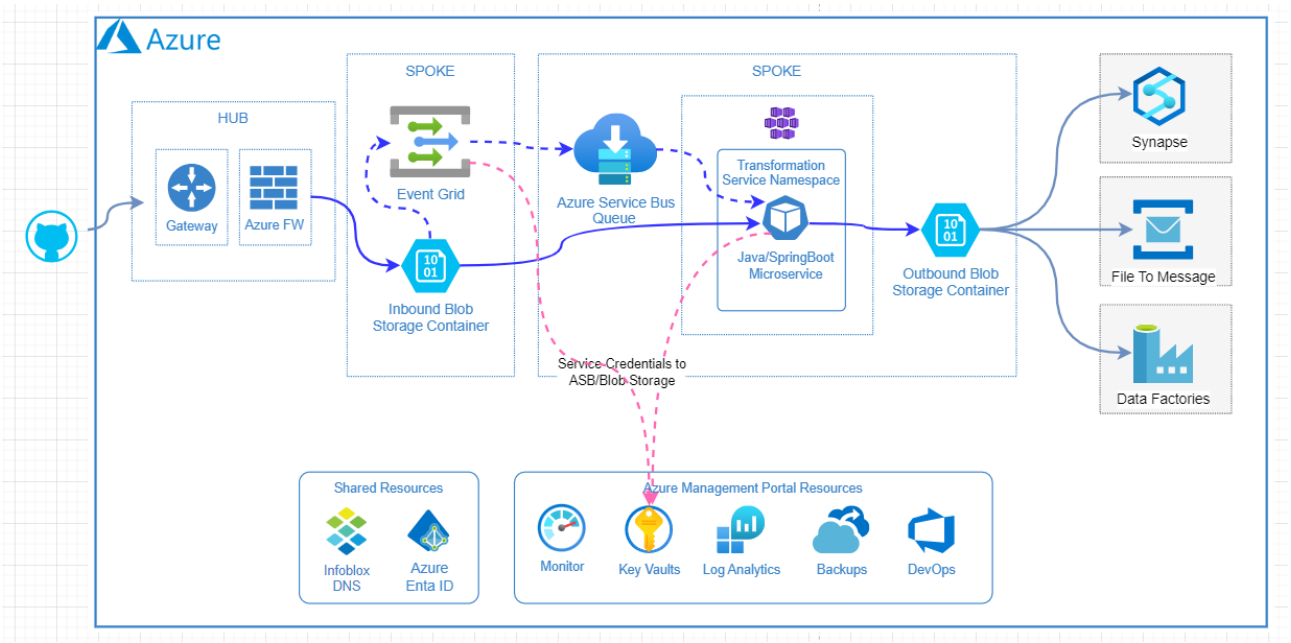

Рисунок 4.1 – Схема архітектури

Основні компоненти системи:

- вихідний код (Code), розробка коду відбувається локально, після чого він зберігається у системі управління версіями;

- DevOps Agent, забезпечує автоматизацію процесів CI/CD, включаючи автоматичне тестування, збірку і розгортання додатків; internal API Management, відповідає за управління API, що включає маршрутизацію запитів, аутентифікацію та авторизацію, а також моніторинг і аналітику використання API;

- API App 1 та API App 2, додатки, які обробляють бізнес-логіку та взаємодіють з базами даних та іншими внутрішніми та зовнішніми службами;
- Azure Application Gateways, використовуються для балансування навантаження та забезпечення додаткових рівнів безпеки між користувачами та додатками в хмарі;
- Azure Private DNS, управління DNS записами в приватному просторі імен, що забезпечує вирішення імен у внутрішній мережі;
- Dev Portal External, зовнішній портал розробника, який забезпечує доступ до документації, інструментів для розробників і спільноти користувачів.

Ця архітектура забезпечує гнучку та масштабовану платформу для розгортання різних додатків та сервісів, дозволяючи легко адаптуватися до змінних вимог бізнесу і забезпечуючи високий рівень безпеки та доступності. Кожен компонент відіграє важливу роль у загальному функціонуванні системи, забезпечуючи виконання вимог до продуктивності, масштабування і надійності.

4.1 Впровадження IaC з використанням Terraform

Під час розробки інфраструктури за допомогою Terraform, велика увага була приділена структуруванню проекту, організації файлів та директорій для забезпечення чіткої архітектури та легкості управління кодом. Це критично важливо для підтримки масштабованості та спрощення подальшого розвитку інфраструктурних проектів.

4.1.1 Організація файлів та директорій

Коренева директорія. У кореневій директорії розміщуються основні конфігураційні файли Terraform, включаючи main.tf, variables.tf, та outputs.tf.

Також тут може знаходитися файл backend.tf для конфігурації backend, який забезпечує зберігання стану та блокування.

main.tf. Головний файл, який включає визначення ресурсів та провайдерів.

variables.tf. Зберігає змінні, які використовуються у конфігурації.

outputs.tf. Визначає вивід, який може використовуватися для отримання важливої інформації про ресурси після розгортання.

Модулі. Використання модулів дозволяє стандартизувати та повторно використовувати код. Для кожного модуля створюється окрема папка всередині директорії modules. Кожен модуль містить свої власні файли main.tf, variables.tf, і outputs.tf.

Лістинг 4.1 – структура модуля

```
/modules
  /network
    main.tf
    variables.tf
    outputs.tf
  /compute
    main.tf
    variables.tf
    outputs.tfmain
```

Середовища. Якщо інфраструктура використовується у кількох середовищах (наприклад, розробка, тестування, продакшн), можна створити окремі конфігураційні файли для кожного середовища всередині папки environments.

Лістинг 4.2 – структура середовищ

```
/environments
  /dev
    main.tf
    terraform.tfvars
  /prod
    main.tf
```

terraform.tfvars файли. Для кожного середовища можуть бути визначені окремі terraform.tfvars файли, що містять значення змінних специфічних для середовища, що дозволяє забезпечити додатковий рівень ізоляції та безпеки конфігурації.

Backend конфігурація. Файл backend.tf налаштовується для кожного середовища для забезпечення зберігання стану Terraform в хмарному сховищі, що підвищує надійність і дозволяє командну роботу над проектом.

4.1.2 Виконання Terraform

Після організації файлів та директорій, процес розгортання полягає в виконанні команд terraform init для ініціалізації проекту, terraform plan для перегляду змін, які будуть застосовані, та terraform apply для застосування змін до середовища. Ці кроки повторюються в кожному середовищі, що забезпечує контрольоване та передбачуване управління інфраструктурою.

4.2 Налаштування та інтеграція компонентів

4.2.1 Azure API Gateway

Azure API Gateway є критично важливим компонентом у хмарній інфраструктурі, оскільки він забезпечує централізоване місце для управління API, включаючи маршрутизацію запитів, безпеку, моніторинг та інші аспекти взаємодії з клієнтами. Ось як можна практично підійти до налаштування Azure API Gateway для оптимальної роботи та інтеграції з іншими компонентами.

Перший крок полягає у створенні інстансу Azure API Management через Azure Portal або за допомогою Terraform. В Terraform це можна зробити за допомогою наступного конфігураційного блоку:

Лістинг 4.3 – створення інстансу Azure API Management

```
resource "azurerm_api_management" "apim" {
  name                = "apim"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  publisher_name      = "apim-dev"
  publisher_email     = "apim@dev.com"

  sku_name = "Developer_1"
}
```

Після створення інстансу, наступні кроки зосереджені на детальній конфігурації API Gateway:

- маршрутизація запитів, визначення правил маршрутизації для різних API, які управляються через Gateway, це включає налаштування базових URL і маршрутів для різних версій API;

- політики безпеки, аутентифікація і авторизація, налаштування політик для перевірки справжності запитів, наприклад, через OAuth2 або інтеграцію з Azure Active Directory;

- обмеження рейту запитів (Rate Limiting), встановлення обмежень на кількість запитів від одного користувача для запобігання зловживань і забезпечення стабільності API;

- моніторинг та логування, використання вбудованих можливостей Azure для збору логів запитів і відповідей; налаштування сповіщень у випадку нештатних ситуацій або перевантаження API.

API Gateway часто взаємодіє з іншими сервісами, такими як Azure Functions або Azure Logic Apps, для обробки бізнес-логіки запитів:

- зв'язок з Azure Functions, інтеграція з Azure Functions може бути налаштована через тригери, що активуються API Gateway, це дозволяє використовувати серверлес обчислення для обробки бізнес-логіки;

- інтеграція з зовнішніми системами, API Gateway може використовуватись для інтеграції з зовнішніми системами, забезпечуючи єдину точку входу для всіх API взаємодій.

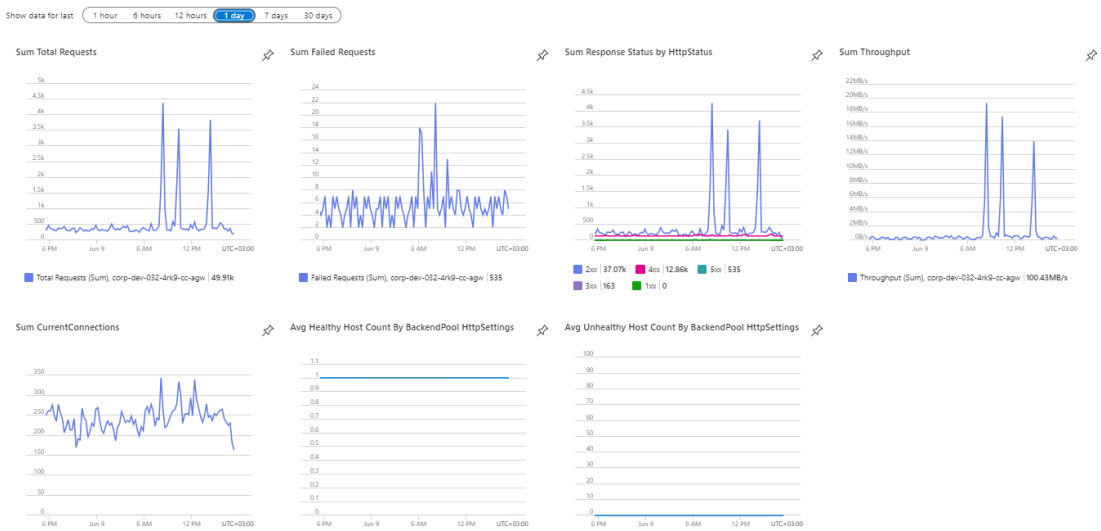

Рисунок 4.2 – Приклад налаштованого API Gateway

Ці кроки не лише підвищують ефективність взаємодії з API, але й забезпечують високий рівень безпеки та надійності системи. Налаштування Azure API Gateway з використанням вищезазначених методів є фундаментальним для створення міцної, безпечної та легко масштабованої хмарної інфраструктури.

4.2.2 Розробка функції в Azure Functions

Для реалізації серверлес обчислень, було створено низку функцій в Azure Functions, які інтегровані з рештою хмарної інфраструктури. Процес розробки включав вибір мови програмування, налаштування виконавчого середовища, кодування бізнес-логіки та її інтеграцію з іншими хмарними сервісами.

Azure Functions було реалізовано на C#, оскільки це забезпечує тісну інтеграцію з іншими сервісами Microsoft Azure та дозволяє легко використовувати існуючі .NET бібліотеки. Функції розроблені для реагування на HTTP запити та події з інших Azure сервісів, таких як Azure Storage Queues.

Процес створення та налаштування.

В Azure Portal було створено нову Function App, де кожна функція має своє власне виконавче середовище та ізольоване сховище коду.

Для кожної функції були налаштовані специфічні для середовища змінні, такі як рядки підключення до баз даних і API ключі, що використовуються для аутентифікації та інтеграції з іншими сервісами.

Було розроблено декілька функцій, зокрема:

- функція обробки HTTP запитів для прийому та відповіді на зовнішні запити;

- функція обробки повідомлень з Azure Queue для асинхронної обробки задач.

Лістинг 4.4 – Створення HTTP тригеру

```
[FunctionName("ProcessHttpRequest")]
public static async Task<IActionResult> Run(
    [HttpTrigger(AuthorizationLevel.Function, "get", "post",
Route = null)] HttpRequest req,
    ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a
request.");

    string name = req.Query["name"];
    string responseMessage = string.IsNullOrEmpty(name)
        ? "This HTTP triggered function executed successfully.
Pass a name in the query string or in the request body for a
personalized response."
        : $"Hello, {name}. This HTTP triggered function executed
successfully.";

    return new OkObjectResult(responseMessage);
}
```

Функції були протестовані на локальному рівні, а потім випробувані в хмарному середовищі за допомогою інструментів, як Postman та Azure Portal.

З використанням Azure Monitor було налаштовано моніторинг виконання функцій, що дозволяє відстежувати продуктивність і вчасно виявляти можливі проблеми. Це забезпечило високу доступність та надійність сервісу.

Розробка та налаштування функцій в Azure Functions дозволила інтегрувати легкі, масштабовані та високоефективні серверлес компоненти в загальну архітектуру системи, значно підвищивши її гнучкість та швидкість відгуку на зміни вимог або навантаження.

4.2.3 Конфігурація баз даних в Azure SQL Database

Azure SQL Database є основним компонентом для зберігання і управління реляційними даними в хмарі. Ефективна конфігурація цього сервісу вимагає детального підходу до налаштування параметрів бази даних, схем безпеки та стратегій резервного копіювання. Ось як було реалізовано практичні аспекти конфігурації Azure SQL Database у проекті:

Використовуючи Azure Portal, було створено нову SQL Database, вказавши назву, регіон, та рівень профілю обслуговування, який забезпечує необхідну продуктивність та масштабованість.

Для автоматизації та управління через код було використано Terraform.

Лістинг 4.5 – Створення HTTP тригеру

```
resource "azurerm_sql_database" " sqldb " {
  name                = "sqldb"
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.sqldb.location
  server_name         = azurerm_sql_server.sqldb.name

  sku_name = "S0"
}
```

Налаштування параметрів, таких як максимальний розмір бази даних, автоматичне збільшення обсягу та кодування.

Безпека та відновлення.

Всі дані в базі шифруються за допомогою Transparent Data Encryption (TDE), щоб забезпечити безпеку даних у спокої.

Налаштування контрольованого доступу до бази даних за допомогою Azure Active Directory та ролей SQL, щоб забезпечити, що тільки авторизовані користувачі мають доступ до чутливих даних.

Автоматичне резервне копіювання було налаштовано для щоденного збереження копій бази даних, що дозволяє відновити дані в разі втрати або пошкодження.

Встановлення політик зберігання резервних копій, щоб гарантувати, що резервні копії доступні протягом визначеного періоду часу та відповідають нормативним вимогам.

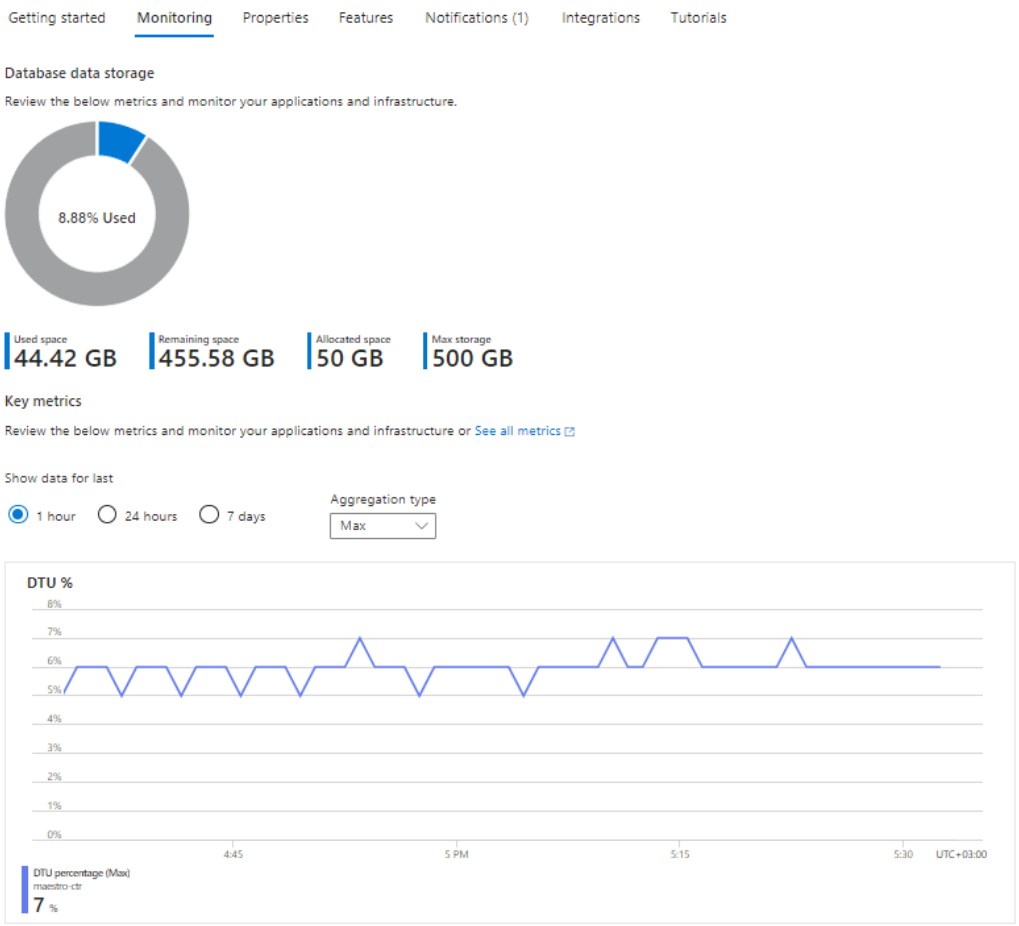

Рисунок 4.3 – Приклад налаштованої Azure SQL Database

4.2.4 Моніторинг та оптимізація

Моніторинг та оптимізація хмарної інфраструктури є ключовими для підтримки стабільної та ефективної роботи системи. В рамках проекту було впроваджено ряд інструментів та практик для відстеження стану інфраструктури та її оптимальної налаштування.

Впровадженно системи моніторингу:

- azure Monitor, використання Azure Monitor дозволило централізовано збирати метрики та логи з усіх компонентів системи, включаючи віртуальні машини, бази даних і Azure Functions, це включало налаштування метрик профілювання продуктивності та використання ресурсів;
- application Insights, для застосунків, які активно використовують Azure Functions і веб-сервіси, було впроваджено Application Insights, який допомагає відслідковувати виконання запитів, виявляти помилки і аналізувати трафік;
- log Analytics, використання Log Analytics дало можливість проводити глибокий аналіз логів та створювати запити для детального аналізу інцидентів, що сприяло швидкому виявленню та усуненню проблем.

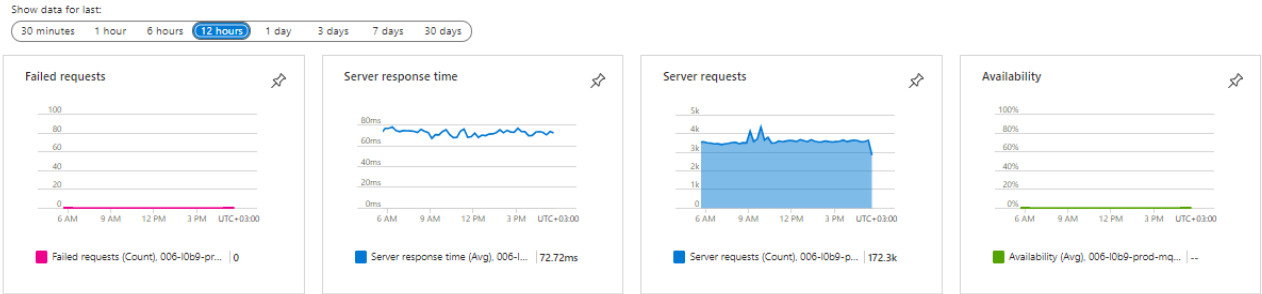

Рисунок 4.4 – Приклад налаштування Application Insights

Також впровадженно системи оптимізації:

- автоматичне масштабування, були налаштовані правила автоматичного масштабування для Azure Functions і віртуальних машин

(VMs), що забезпечувало динамічне збільшення або зменшення кількості ресурсів залежно від поточного навантаження, це допомогло оптимізувати витрати та підвищити продуктивність системи;

- оптимізація SQL запитів, в Azure SQL Database було реалізовано процедури для оптимізації SQL запитів, зокрема налаштування індексів та перегляд планів виконання для мінімізації часу відповіді бази даних;

- покращення безпеки, налаштування політик безпеки для забезпечення захисту даних і ресурсів, впровадження шифрування даних у транзиті та спокої, налаштування систем ідентифікації і автентифікації, і використання Azure Security Center для автоматичного аудиту та рекомендацій щодо покращення безпеки.

4.2.5 Звітність і сповіщення

У рамках проекту було розроблено систему звітності та сповіщень, що дозволяє забезпечити ефективне моніторинг і управління хмарною інфраструктурою. Важливим аспектом цього підходу є не тільки реакція на потенційні проблеми, а й проактивне управління ресурсами та оптимізація витрат.

Налаштовано автоматичні сповіщення, які активуються при досягненні критичних метрик або виявленні помилок в системі. Це дозволяє команді оперативно реагувати на проблеми, забезпечуючи надійність і стабільність системи. Сповіщення реалізовані через електронні листи та SMS-повідомлення, забезпечуючи широке охоплення та швидке інформування відповідальних осіб.

Система автоматично генерує та розсилає щотижневі та щомісячні звіти зацікавленим сторонам. Звіти містять аналіз продуктивності, використання ресурсів та ефективності витрат. Це забезпечує постійний моніторинг важливих показників системи і допомагає виявляти тренди та оптимізувати процеси.

Звіти включають графіки, діаграми та інші візуальні засоби представлення даних, що сприяє кращому сприйняттю і аналізу інформації. Використання інструментів візуалізації, таких як Microsoft Power BI, дозволяє детально розглядати великі обсяги даних та виводити цінні бізнес-інсайти.

Завдяки реалізації вдосконаленої системи звітності та сповіщень, проект досяг значних успіхів у підвищенні ефективності роботи хмарної інфраструктури. Автоматизація процесів звітності та моніторингу дозволила оперативно реагувати на зміни, оптимізувати ресурси та покращити загальну продуктивність системи. Це не тільки забезпечило стабільність та високу доступність сервісів, але й сприяло зниженню оперативних витрат, що є важливим фактором у забезпеченні бізнес-конкурентоспроможності.

4.2.6 Тестування та валідація

Ефективне тестування та валідація системи є критично важливими для забезпечення її надійності, продуктивності та безпеки. Розробка тестових сценаріїв дозволяє систематично перевіряти кожен компонент інфраструктури та взаємодію між ними, виявляючи потенційні проблеми на ранніх етапах розробки та впровадження.

Процес розробки тестових сценаріїв:

- визначення цілей тестування, основні цілі включають перевірку функціональності, продуктивності, безпеки та витривалості системи, це допомагає забезпечити, що система відповідає всім технічним та бізнес-вимогам;

- розробка функціональних тестів, інтеграційні тести, перевірка взаємодій між компонентами системи, такими як бази даних, API Gateway, та серверлес функції;

- кінцеві тести, симуляція реальних користувацьких сценаріїв для перевірки системи як єдиного цілого;

- розробка тестів продуктивності, навантажувальні тести, імітація пікових навантажень для визначення відповідності системи вимогам до продуктивності;

- стрес-тести, виявлення меж продуктивності системи шляхом створення екстремальних умов роботи; розробка тестів безпеки, тести на проникнення, імітація атак на систему для перевірки захисту від зловмисників;

- тести на вразливості, використання автоматизованих інструментів для сканування системи на предмет відомих вразливостей;

- використання інструментів, таких як Selenium для веб-інтерфейсів, JMeter для тестування API, Azure DevOps для організації CI/CD пайплайнів, це забезпечує неперервне тестування та виявлення проблем на ранніх етапах розробки.

Для функції, що обробляє HTTP запити, можна розробити наступний тестовий сценарій

Лістинг 4.6 – тестовий сценарій

```
import requests

def test_azure_function():
    url = "http://<your-function-url>?name=Test"
    response = requests.get(url)
    assert response.status_code == 200
    assert response.json() == {"message": "Hello, Test. This
HTTP triggered function executed successfully."}
```

Цей тест перевіряє, що функція правильно реагує на HTTP GET запит і повертає коректну відповідь.

Після розробки тестів вони регулярно запускаються в процесі розробки та після кожного оновлення системи. Результати тестів аналізуються для виявлення проблем, що вимагають уваги, і вживаються заходи для їх виправлення. Це забезпечує постійне покращення якості продукту та високий рівень задоволення користувачів.

4.3 Впровадження GitHub Actions для CI/CD процесів

У рамках проекту було успішно впроваджено GitHub Actions для автоматизації процесів безперервної інтеграції (CI) та безперервної доставки (CD), що забезпечило високий рівень ефективності управління кодом і інфраструктурою. Ця інтеграція дозволила автоматизувати тестування, розгортання та моніторинг, значно підвищуючи швидкість внесення змін та знижуючи ризик помилок.

Реалізовані процеси через GitHub Actions:

- налаштування автоматичного запуску тестів при кожному коміті у головну гілку, це гарантує, що всі зміни перевіряються на помилки перед їх інтеграцією в основну кодову базу, сприяючи підтримці високої якості коду;
- автоматичне розгортання змін на тестові та продакшн середовища після проходження всіх тестів і перевірок, це дозволяє забезпечити безпечне і швидке впровадження нових функцій та виправлень, мінімізуючи простої і вплив на кінцевих користувачів;
- використання GitHub Actions для виконання terraform plan і terraform apply забезпечує контрольоване і відтворюване впровадження інфраструктурних змін, це важливо для забезпечення консистенції інфраструктури та уникнення помилок, пов'язаних з ручним впровадженням;
- всі чутливі дані, такі як ключі доступу та конфігураційні параметри, зберігаються у секретах GitHub, що забезпечує їх безпеку та унеможливлює несанкціонований доступ, використання засобів контролю доступу і рецензування коду перед злиттям змін у головну гілку допомагає підтримувати високі стандарти якості та безпеки.

Візуалізація процесів.

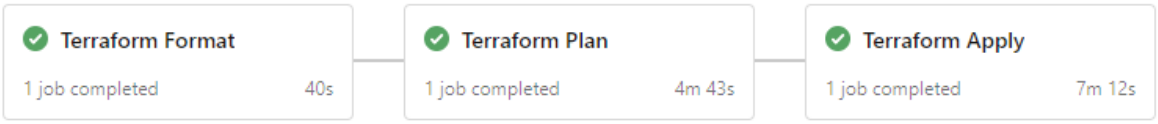

Рисунок 4.5 – Результат виконання GitHub Actions

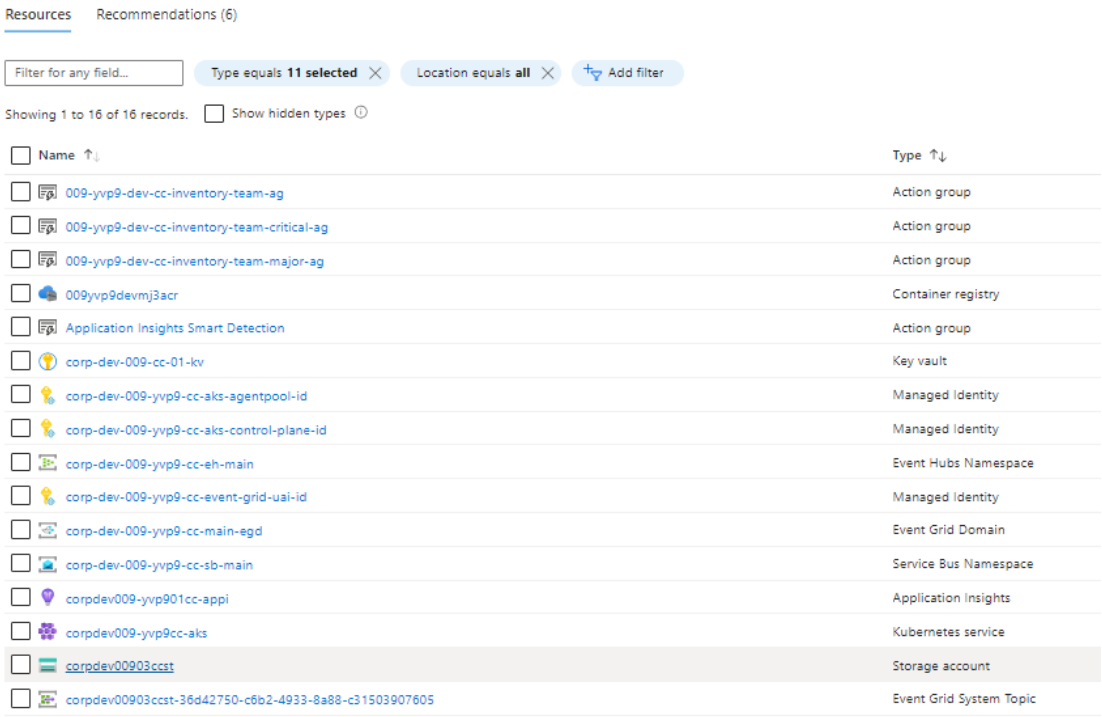

Рисунок 4.6 – Створена інфраструктура після виконання GitHub Actions

Це візуалізація демонструє як GitHub Actions автоматизує процеси CI/CD, включаючи перевірку коду, тестування, планування змін інфраструктури та їх впровадження, що забезпечує високу ефективність та надійність системи.

4.4 Оцінка продуктивності, надійності та економічної ефективності

Після впровадження та тестування системи, наступний важливий етап полягає в оцінці її продуктивності, надійності та економічної ефективності.

Цей аналіз допомагає визначити, чи відповідає розроблена інфраструктура поставленим бізнес-вимогам та чи є вона ефективною з точки зору витрат.

Методи оцінки продуктивності:

- метрики продуктивності, вимірювання часу відгуку системи на запити користувачів, включаючи час обробки на сервері та час передачі даних; пропускна здатність, оцінка кількості операцій, які система може обробити за одиницю часу, зокрема запити до баз даних і транзакції;
- використання ресурсів, аналіз використання CPU, пам'яті, дискового простору та мережевих ресурсів для визначення потенційних "вузьких місць" в системі;
- навантажувальні та стрес-тести, проведення навантажувальних тестів для визначення продуктивності системи при різних рівнях навантаження та стрес-тестів для визначення її поведінки в крайніх умовах.

Оцінка надійності:

- час безвідмовної роботи (Uptime), моніторинг часу безвідмовної роботи системи для визначення її стабільності та доступності;
- відновлення після збоїв, тестування процесів відновлення системи після планових або несподіваних збоїв для перевірки ефективності стратегій відновлення;
- резервне копіювання та аварійне відновлення, оцінка стратегій та практик резервного копіювання та відновлення даних для гарантії цілісності даних та швидкого відновлення роботи системи.

Оцінка економічної ефективності:

- витрати на інфраструктуру, аналіз загальних витрат на хмарну інфраструктуру, включаючи витрати на обчислювальні ресурси, зберігання даних, мережеві послуги та ліцензування;
- оптимізація витрат, використання інструментів для управління хмарними витратами, таких як Azure Cost Management, для ідентифікації можливостей зниження витрат через оптимізацію ресурсів;

- ROI (Return on Investment), розрахунок повернення інвестицій з урахуванням загальних витрат на проект та отриманих вигод від впровадження системи, таких як підвищення продуктивності, зниження операційних витрат та покращення якості обслуговування користувачів.

Таблиця 4.1 – порівняння витрат з класичним підхідом та через Terraform

Критерій	Класичний підхід	Автоматизація через Terraform
Людино-години на налаштування	100 годин	30 годин
Вартість налаштування	$5,000 (100 годин × $50/година)	$1,500 (30 годин × $50/година)
Частота оновлень на рік	12 разів	12 разів
Людино-години на оновлення	10 годин на оновлення × 12 = 120 годин	2 години на оновлення × 12 = 24 години
Вартість оновлень	$6,000 (120 годин × $50/година)	$1,200 (24 години × $50/година)
Загальна річна вартість	$11,000	$2,700

Ці методи дозволяють не тільки визначити, наскільки ефективно система виконує свої функції, але й оцінити її здатність адаптуватися до зростаючих вимог та оптимізувати витрати для забезпечення максимальної вигоди від інвестицій в хмарні технології.

ВИСНОВКИ

Впровадження інноваційних методів оптимізації розподілу ресурсів у масштабованих хмарних середовищах, зокрема на базі Azure Cloud та за допомогою Terraform, дозволило відкрити нові горизонти для підвищення ефективності, надійності та економічності хмарних інфраструктур. Наукова новизна даної роботи полягає у розробці модифікованої метода автоматизації, яка, на відміну від існуючих підходів, включає інтелектуальне масштабування та декларативний опис інфраструктури, що сприяє більш ефективному задоволенню змінних потреб сучасних додатків та сервісів.

Основні висновки роботи:

- автоматизація інфраструктури за допомогою Terraform значно спрощує управління хмарними ресурсами, забезпечує високу швидкість внесення змін та підтримку великої кількості середовищ і конфігурацій, водночас знижуючи ризик людських помилок;

- динамічне масштабування в Azure Cloud, зокрема використання Azure Autoscale, дозволяє ефективно реагувати на зміни навантаження, оптимізуючи використання ресурсів і витрати на хмарні послуги, при цьому підтримуючи необхідний рівень продуктивності та доступності сервісів;

- інтеграція інструментів і платформ, таких як Terraform і Azure, відкриває додаткові можливості для більш гнучкого управління хмарною інфраструктурою, дозволяючи компаніям швидко адаптуватися до змін на ринку та технологічних інновацій.

Вплив оптимізації на економічну ефективність та продуктивність Аналіз витрат і продуктивності до та після оптимізації яскраво демонструє, що ефективне управління хмарними ресурсами може призвести до значного зниження витрат – приблизно на 30-35% від початкових значень, а також до підвищення продуктивності сервісів на 10-15%, що можна побачити в таблиці.

Загалом, застосування сучасних підходів до управління хмарними ресурсами, включаючи Terraform для декларативного опису інфраструктури та інструменти автомасштабування в Azure, визначило нові стандарти у побудові ефективних, гнучких та економічно вигідних хмарних рішень. Такий підхід дозволяє компаніям не тільки знижувати витрати, але й забезпечувати вищу якість наданих хмарних сервісів, а також сприяє стійкому розвитку та екологічній відповідальності в індустрії.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wang, D., Zhong, D. and Li, L. "A comprehensive study of the role of cloud computing on the information technology infrastructure library (ITIL) processes", Library Hi Tech, (2022), Vol. 40 No. 6, pp. 1954-1975. DOI: https://doi.org/10.1108/LHT-01-2021-0031

2. Omar Alzakholi, Lailan M. Haij, Hanan M. Shukur, Rizgar R. Zebari, Shakir M. Abas, Mohammad A. M. Sadeeq Comparison Among Cloud Technologies and Cloud Performance (2020) Vol. 1 No. 1 DOI: https://doi.org/10.38094/jastt1219

3. Рудь Л. І. Войцеховська О. В. ВИКОРИСТАННЯ ХМАРНИХ ТЕХНОЛОГІЙ В МЕТОДОЛОГІЇ DEVOPS ТА CI/CD ПРОЦЕСІ. 2021 No УДК 004.056 13 - 15 DOI: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/34093/89359.pdf?sequence=2&isAllowed=y

4. Коломицев М. В. Підхід до побудови системи безпеки хмарних баз даних Johnson та Smith 2020 No УДК 004.7. 2-3 DOI: https://ela.kpi.ua/server/api/core/bitstreams/d740a6c4-936d-4d95-a2e5-b41c5686a75e/content

5. Sururah A. Bello, Lukumon O. Oyedele, Olugbenga O. Akinade, Muhammad Bilal, Juan Manuel Davila Delgado, Lukman A. Akanbi, Anuoluwapo O. Ajayi, Hakeem A. Owolabi Review Cloud computing in construction industry: Use cases, benefits and challenges 2020 Vol. 6 No. 3, pp. 54-60 DOI: https://doi.org/10.1016/j.autcon.2020.103441

6. Mohammad S. Aslanpour a b, Sukhpal Singh Gill c, Adel N. Toosi Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research 2020 Vol. 40 No. 6, pp. 54-75. DOI: https://doi.org/10.1016/j.iot.2020.100273

7. Кулик В.В. Дослідження методів оптимізації обчислень у хмарних технологіях 2020 12 – 15 DOI: https://openarchive.nure.ua/server/api/core/

bitstreams/c0243f1e-34a0-42e0-b127-d87f104e6c43/content

8. Ількевич Н.С. Хмарні технології в освіті 2021 No УДК 004:37 27 – 29 DOI: http://eprints.zu.edu.ua/33187/1/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0_%D0%A5%D0%BC%D0%B0%D1%80%D0%BD%D1%96%20%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%97.pdf

9. Сапсай Т.Г. Система розподілу навантаження на сервіси хмарної інфраструктури 2018 No УДК 004.75 25 – 27 DOI: https://ela.kpi.ua/server/api/core/bitstreams/32b16060-dc04-4a6a-bc54-b6c8cf7331ed/content

10. Гуржій В. В. ТЕОРЕТИЧНІ АСПЕКТИ ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ В УПРАВЛІННІ ПРОЄКТАМИ 2023 No УДК 005:004.8 4 - 5 DOI: https://doi.org/10.32702/2307-2105.2023.12.73

11. Огляд служб оптимізації виконання й повернення замовлень 2024 https://learn.microsoft.com/uk-ua/dynamics365/intelligent-order-management/fulfillment-returns-optimization

12. Microsoft Cost Management 2024 https://azure.microsoft.com/en-us/products/cost-management

13. Волк М.О., Поповкін М.М. Методи моделювання масштабованих ресурсів. Системи управління, навігації та зв’язку. № 3. 2024. С. 56-60. УДК 004. 7:004.415-025.25 56 – 60 DOI: 10.26906/SUNZ.2024.3.056