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 та засоби автоматичного розгортання IaC для хмарних сервісів

(тема)

Виконав:

студент II курсу, групи СПм-22-2

Копцев О.О
(прізвище, ініціали)

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

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

Освітня програма Системне програмування
(повна назва освітньої програми)

Керівник: проф. Мартовицький В.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-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Копцеву Олегу Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи та засоби автоматичного розгортання IaC для хмарних сервісів

затверджена наказом по університету від " 06 " листопада 2023 р. № 1299 Ст

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

3. Вхідні дані до роботи 1) документація Azure Resource Manager Templates та Bicep; 2) офіційна документація та репозиторії модулів Terraform; 3) специфікації та документація AWS CloudFormation; 4) Інтегровані середовища розробки Visual Studio Code; 5) документація Google Cloud Deployment Manager 6) документація та приклади скриптів на базі Chef для управління інфраструктурою.

4. Перелік питань, що потрібно опрацювати у роботі
1) огляд та аналіз сучасних методів автоматичного розгортання IaC
2) детальний аналіз інструментів для розгортання IaC
3) розробка та обгрунтування методики дослідження
4) практична реалізація моделей автоматичного розгортання
5) експериментальні дослідження та аналіз результатів
6) висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 8 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	Огляд протоколів корпоративних комп'ютер-	07.11.23-13.11.23	
	них мереж		
2	Вибір та обгрунтування методики дослідження	14.11.23-20.11.23	
3	Вибір інструментальних засобів	21.11.23-23.11.23	
4	Розробка моделей протоколів	24.11.23-06.12.23	
5	Проведення експериментів	07.12.23-23.12.23	
6	Оформлення матеріалів кваліфікаційної роботи	26.12.23-02.01.24	
7	Подання кваліфікаційної роботи керівникові та	03.01.24-06.01.24	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	09.01.24-12.01.24	
	вання		

Дата видачі завдання 06 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Мартовицький В.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 62 с., 2 рис., 2 табл., 1 дод., 12 джерел.

ІНФРАСТРУКТУРА ЯК КОД, ХМАРНІ СЕРВІСИ, АВТОМАТИЧНЕ РОЗГОРТАННЯ, ШАБЛОНИ, ДОМЕННО-ОРІЄНТОВАНА МОВА, ДЕКЛАРАТИВНИЙ КОД, МАСШТАБОВАНІСТЬ, AZURE, ARM TEMPLATES.

Метою кваліфікаційної роботи є дослідження методів та засобів автоматичного розгортання інфраструктури як коду для хмарних сервісів.

У ході виконання кваліфікаційної роботи для реалізації поставленої мети були досліджені основні методи розгортання інфраструктури та відомі інструменти автоматичного розгортання інфраструктури як коду провідних провайдерів хмарних платформ, а саме: AWS CloudFormation, Azure Resource Manager Templates, Google Cloud Deployment Manager, IBM Cloud Schematics та Oracle Cloud Infrastructure Resource Manager. Для практичної реалізації завдання автоматичного розгортання інфраструктури як коду були обрані засоби, що мають найбільшу кількість функціональних можливостей IaC такі як Azure Resource Manager Templates, доменно-орієнтована мова Вісер та мультиплатформний Terraform.

ABSTRACT

Master’s thesis: 62 pages, 2 figures, 2 tables, 1 appendices, 12 sources.

INFRASTRUCTURE AS CODE, CLOUD SERVICES, AUTOMATED DEPLOYMENT, TEMPLATES, DOMAIN-SPECIFIC LANGUAGE, DECLARATIVE CODE, SCALING, AZURE, ARM TEMPLATES.

The major goal of this thesis is to research methods and means of automatic deployment of infrastructure as a code for cloud services.

To realize the goal, the main infrastructure deployment methods and well-known tools for automatic infrastructure deployment as code from leading cloud platform providers were investigated, namely: AWS CloudFormation, Azure Resource Manager Templates, Google Cloud Deployment Manager, IBM Cloud Schematics, and Oracle Cloud Infrastructure Resource Manager. For the practical implementation of the task of automatic deployment of infrastructure as a code, tools with the largest number of IaS functionalities were chosen, such as Azure Resource Manager Templates, the domain-oriented language Bicep, and the multi-platform Terraform.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ARM Templates - Azure Resource Manager Templates

AWS - Amazon Web Services

DSL - доменно-орієнтована мова (Domain-Specific Language)

IaC - Інфраструктура як код (Infrastructure as a Code)

GCP - Google Cloud Platform

OCI - Oracle Cloud Infrastructure

ВСТУП

Коли термін «хмара» вперше з’явився на початку 2000-х років, він був оточений містикою. Ідея доступу до обчислювальних ресурсів ззовні локальної IT-інфраструктури (з неба?) сприймалася як наукова фантастика. Але реальність виявилася набагато глибшою і назавжди змінила технології та методи ведення бізнесу.

До ери хмарних обчислень організації купували та підтримували локальну IT-інфраструктуру. Незважаючи на те, що економія коштів була великою рушійною силою переходу до хмари, багато організацій вважають, що хмарна інфраструктура має багато переваг.

Хмарні обчислення надають передові обчислювальні ресурси, які доступні за вимогою, масштабуються за вимогою та отримують регулярні оновлення без необхідності купувати та підтримувати локальну інфраструктуру. Завдяки хмарним обчисленням команди стають більш ефективними та скорочують час виходу на ринок, оскільки вони можуть швидко шукати та масштабувати послуги без значних зусиль, необхідних для керування традиційною локальною інфраструктурою.

Розвиток апаратної віртуалізації в середині 2000-х створив нові можливості для інфраструктури хмарного хостингу. Провайдери хмарного хостингу почали надавати доступ до платформ динамічної хмарної інфраструктури. Необхідність швидкого налаштування та керування складною хмарною інфраструктурою швидко стала серйозною проблемою.

Ідея інфраструктури як коду (IaC) або моделювання інфраструктури за допомогою коду натхненна успіхом постійної інтеграції та безперервної доставки. Принцип автоматизації робочих процесів розробки програмного забезпечення також допоміг в адмініструванні хмарних систем. Без IaC керування інфраструктурою вручну відбувається повільно. Якщо зміни в інфраструктурі потрібні через аномалії навколишнього середовища,

зростання трафіку чи інші проблеми, неможливо сказати, скільки часу знадобиться, щоб відреагувати та внести виправлення. Завдяки IaC інфраструктура автоматично адаптується до змін конфігурації та реагує на збільшення трафіку за допомогою автоматичного масштабування.

IaC-обробка - це спосіб управління конфігурацією, коли інфраструктурні ресурси організації зашифровуються в текстових файлах. Для застосування цього підходу потрібна низка залежностей, наприклад, платформи хостингу та інструменти автоматизації, тобто, інструменти автоматичного розгортання інфраструктури як коду, що сьогодні надаються постачальниками хмарних платформ.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні терміни та поняття хмарних обчислень

Хмарні обчислення - це модель надання та доступу до різноманітних обчислювальних ресурсів (таких як сервери, сховища, мережі, програмне забезпечення) через інтернет. У цій моделі ви не володієте фізичним обладнанням, а використовуєте ресурси, що господарюються у великих дата-центрах (хмарах), під управлінням постачальників хмарних послуг.

Постачальники хмарних послуг - це компанії або організації, які надають доступ до різноманітних обчислювальних, мережевих та інших інформаційних ресурсів через інтернет. Ці компанії мають великі дата-центри з великою кількістю обладнання, що використовується для надання послуг у моделі хмарних обчислень. Ось деякі провідні постачальники хмарних послуг:

- Amazon Web Services є одним з найбільших та найпопулярніших постачальників хмарних послуг у світі. Надає широкий спектр послуг, включаючи обчислювальні потужності, зберігання, бази даних, штучний інтелект, інтернет речей (IoT), аналітику та багато іншого. Має глобальну мережу дата-центрів, що дозволяє користувачам розгортати свої застосунки в різних регіонах світу;

- Microsoft Azure є популярним постачальником хмарних послуг від Microsoft. Надає широкий спектр послуг, включаючи обчислювальні ресурси, аналітику, штучний інтелект, інтернет речей, адміністрування, зберігання та інше. Інтегрується з іншими продуктами Microsoft, що дозволяє легше розгортати та управляти застосунками підприємств;

- Google Cloud Platform — це обширний набір хмарних послуг, що надається компанією Google. Він пропонує широкий спектр послуг, які включають обчислювальні ресурси, зберігання даних, машинне навчання,

аналітику даних, інтернет речей та багато іншого. GCP славиться своєю інноваційністю, особливо в сферах машинного навчання та аналітики великих даних, і інтегрується з іншими сервісами та продуктами Google, що надає додаткові переваги для розробників і підприємств;

- IBM Cloud — це хмарний сервіс, що надається компанією IBM. Він пропонує різноманітні хмарні рішення, включаючи інфраструктуру як сервіс (IaaS), платформу як сервіс (PaaS) та програмне забезпечення як сервіс (SaaS). IBM Cloud зосереджується на підтримці складних підприємницьких додатків і сервісів, надаючи потужні інструменти для штучного інтелекту, машинного навчання, інтернету речей, блокчейна та багато іншого, інтегруючись з широким спектром технологій та сервісів IBM.

- Alibaba Cloud, також відомий як Aliyun, є провідним хмарним сервісом у Китаї, наданим Alibaba Group. Він пропонує широкий спектр послуг, включаючи обчислювальні можливості, зберігання даних, реляційні та NoSQL бази даних, великі дані, аналітику, машинне навчання та багато іншого. Alibaba Cloud особливо популярний у Азії і є важливим вибором для компаній, які шукають хмарні рішення в цьому регіоні;

- Oracle Cloud надається компанією Oracle і пропонує різноманітні хмарні сервіси, включаючи обчислювальні ресурси, зберігання даних, бази даних як сервіс, а також платформи для розробки та розгортання додатків. Він особливо знаменитий своїми продуктами баз даних і пропонує потужні рішення для управління даними та аналітики. Oracle Cloud є важливим вибором для підприємств, які шукають інтегровані хмарні рішення, особливо для тих, хто вже використовує продукти Oracle.

Веб-сервіси Amazon Web Services, Microsoft Azure і Google Cloud Platform є трьома великими постачальниками хмарних послуг сьогодні. Разом вони займають 66% світового ринку хмарної інфраструктури, що більше, ніж 63% у попередньому році, згідно з даними Synergy Research Group (рисунок 1.1) [1].

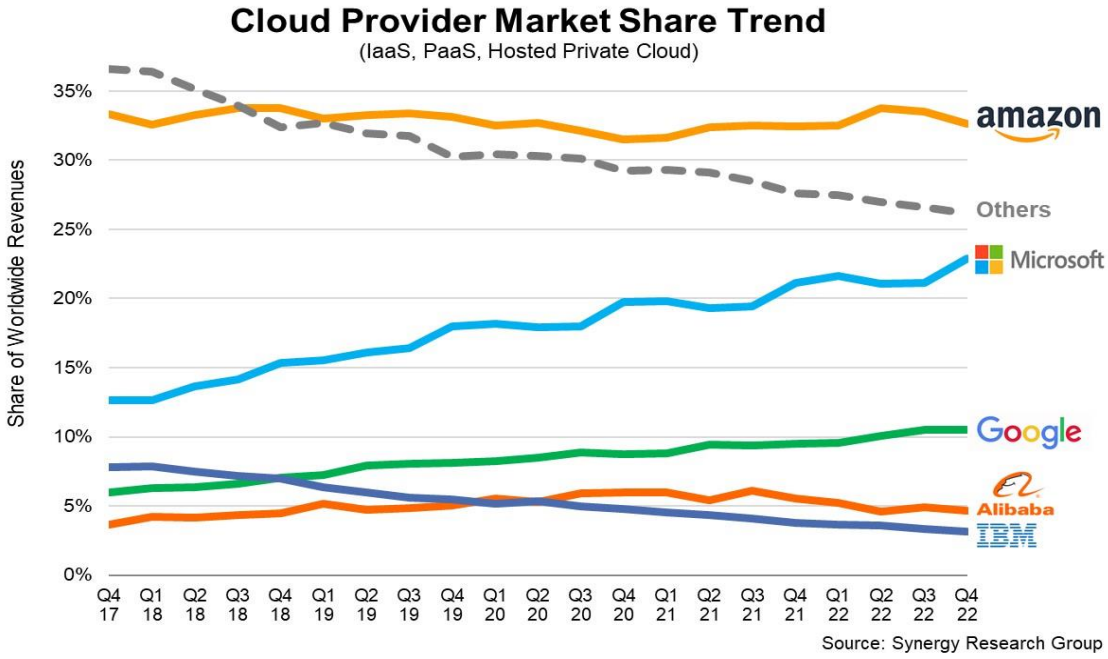

Рисунок 1.1 - Частка ринку постачальників хмарних послуг на початок 2023 року

Багато провайдерів хмарних послуг мають свої центри даних по всьому світу, що забезпечує глобальну доступність та надійність послуг. Кожен постачальник має свої особливості, переваги та специфіку послуг. Вибір постачальника хмарних сервісів залежить від потреб бізнесу, технічних вимог та бюджету. Багато організацій використовують комбінацію послуг від різних постачальників для оптимізації своїх бізнес-процесів.

Хмарні сервіси, відомі також як хмарні послуги - це послуги та ресурси, які надаються через інтернет (хмару) для забезпечення доступу до обчислювальних, мережевих та інших IT-ресурсів без необхідності фізичного володіння та керування обладнанням.

1.2 Основні характеристики хмарних сервісів

Компанії часто уникають створення власної хмарної інфраструктури, оскільки це дорого, потребує великого досвіду та постійного обслуговування.

Багато компаній вважають за краще не займатися всім цим, особливо тому, що використання надійного постачальника хмарних послуг забезпечує більше переваг, таких як:

- доступ через інтернет (availability). Хмарні сервіси надаються та доступні через інтернет. Користувачі можуть отримати доступ до послуг з будь-якого місця та пристрою, що має підключення до мережі;

- еластичність та масштабованість (elasticity & scalability) для користувача означає те, що вони можуть масштабувати ресурси вгору або вниз в залежності від потреб. Це дозволяє ефективно використовувати ресурси та оптимізувати витрати;

- платіж за використання (pay-as-you-go). Модель оплати за використання ресурсів. Користувачі платять лише за те, що вони фактично використовують, що дозволяє економити кошти та оптимізувати бюджет;

- самообслуговування (self-service) полягає в тому, що користувачі можуть самостійно налаштовувати та управляти своїми ресурсами, не чекаючи на інші органи адміністрації.

Хмарні сервіси включають в себе різноманітні послуги, такі як обчислювальні потужності (віртуальні машини), сховища даних, мережеві сервіси, аналітика, штучний інтелект, інтернет речей (IoT) та інші.

Розглянемо основні моделі та характеристики хмарних сервісів (рисунок 1.2) [3]:

- інфраструктура як послуга (IaaS). Цей сервіс надає можливість орендувати віртуальні машини, мережеві пристрої, сховища тощо. Ви самі відповідаєте за управління операційними системами, додатками та даними на цих віртуальних машинах;

- платформа як послуга (PaaS). Надає середовище для розробки та виконання програм безпосередньо в хмарі. Постачальник бере на себе управління інфраструктурою, мережами, операційними системами та оновленнями, ви фокусуєтеся на розробці програм;

- програмне забезпечення як послуга (SaaS). Цей сервіс надає готові до

використання додатки та сервіси через мережу. Наприклад, електронна пошта, офісні пакети, CRM-системи тощо. Користувачі не керують інфраструктурою та мають доступ тільки до готових програм через веб-браузер.

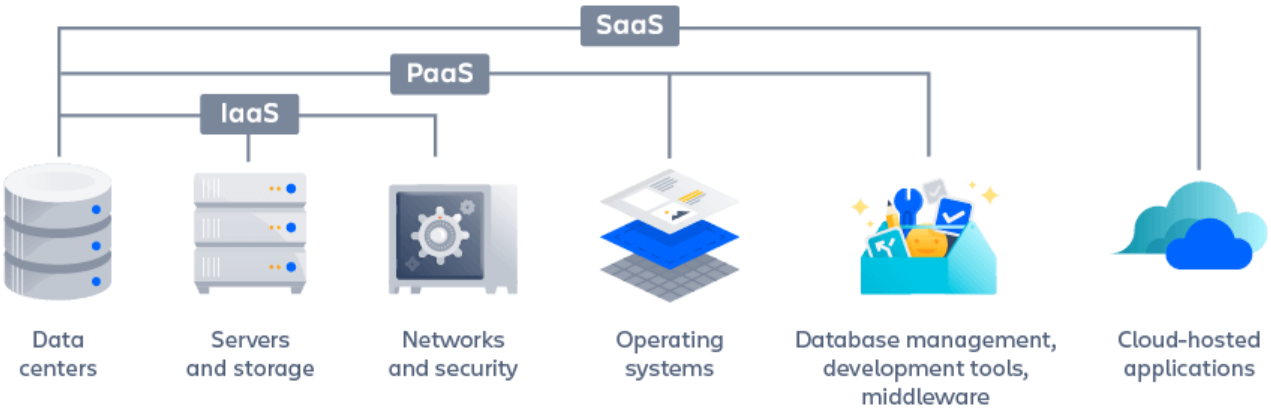

Рисунок 1.2 - Сервіси хмарних обчислень

Поза основними моделями хмарних послуг (IaaS, PaaS, SaaS), існує кілька інших видів послуг, які можуть бути надані постачальниками хмарних послуг. Ось кілька з них:

- FaaS (Functions as a Service). Це модель, де розробники можуть розгортати та виконувати окремі функції або фрагменти коду без необхідності керування інфраструктурою. Платять лише за викликані та виконані функції;

- CaaS (Containers as a Service). Це сервіс, який надає можливість розгортати, керувати та оркеструвати контейнери, використовуючи певну оркестраційну систему (наприклад, Kubernetes). Дозволяє швидко запускати та масштабувати додатки, упаковані у контейнери;

- DaaS (Data as a Service). Цей тип послуги надає доступ до різних даних та їхнього аналізу через інтернет. Включає послуги аналітики даних, бази даних, потокову обробку даних тощо;

- DRaaS (Disaster Recovery as a Service). Це послуга, яка надає

можливість планування та відновлення роботи системи в разі надзвичайних ситуацій або аварій. Забезпечує надійність та відновлюваність даних та додатків;

- BaaS (Backend as a Service). Це послуга, яка надає можливість використовувати готові серверні функції та служби для розробки додатків, спрощуючи розробку бекенду;

- IoTaaS (Internet of Things as a Service). Цей тип послуги дозволяє підключати та керувати пристроями інтернету речей та обробляти їхні дані в хмарному середовищі.

Кожен з цих типів послуг спеціалізується на конкретній області та надає можливості для розширення та оптимізації додатків та бізнес-процесів. Хмарні сервіси дозволяють підприємствам та користувачам миттєво користуватися потужністю та функціональністю IT-ресурсів без значних витрат на обладнання та управління ним та забезпечують доступ до ресурсів через інтернет з будь-якого місця та пристрою. Тому, спільний доступ (Ubiquitous Access) також є однією з основних характеристик хмарних сервісів.

1.3 Основні компоненти інфраструктури хмарних сервісів

Всі хмарні сервіси використовують певну інфраструктуру для надання своїх послуг [2]. Ця інфраструктура складається з різних компонентів та технологій, які дозволяють забезпечувати доступність, масштабованість, безпеку та інші характеристики хмарних сервісів. Основні складові інфраструктури хмарних сервісів включають наступні елементи:

- сервери: ядро хмарних сервісів, що забезпечують потужні обчислювальні ресурси для виконання широкого спектру задач;

- сховища даних: віртуальні репозиторії для збереження великих обсягів даних, які можуть бути легко масштабовані та доступні згідно з потребами користувачів;

- мережева інфраструктура: включає комутатори, маршрутизатори, файрволи та інше необхідне обладнання для забезпечення стабільного мережевого з'єднання та ефективної комунікації між усіми компонентами хмари;

- технології віртуалізації: забезпечують оптимальне використання обчислювальних ресурсів, дозволяючи кільком користувачам ефективно працювати на спільному фізичному обладнанні;

- інструменти управління: набори інструментів для моніторингу, управління та оптимізації ресурсів, що використовуються у хмарних сервісах;

- безпека: включає різноманітні механізми для захисту даних, мереж та інфраструктури від несанкціонованого доступу та інших загроз;

- технології розгортання та оркестрації контейнерів: інструменти для ефективного розгортання, управління та оркестрації контейнеризованих додатків;

- інфраструктура для мікросервісів: надає оптимальні умови для розгортання та управління архітектурою мікросервісів;

- додаткові сервіси та компоненти: включає унікальні хмарні рішення, що пропонуються певними провайдерами, такі як інструменти штучного інтелекту, інтеграційні платформи, мобільні сервіси та інше.

1.4 Методологія автоматизації та управління інфраструктурними ресурсами

Інфраструктура як код (IaC) не є хмарною послугою самою по собі, але вона часто використовується у хмарних оточеннях та сприяє оптимізації управління та розгортанням хмарних послуг [4].

Інфраструктура як код - це методологія автоматизації та управління інфраструктурними ресурсами, при якому вся інфраструктура (сервери, мережі, сховища тощо) описується та управляється за допомогою

програмного коду. Це означає, що можна описати інфраструктуру у вигляді коду (наприклад, за допомогою мови програмування або спеціалізованої доменно-орієнтованої мови), а потім автоматизовано розгортати та керувати нею.

Цей підхід є дуже популярним у хмарних середовищах, оскільки дозволяє швидко та ефективно налаштовувати, масштабувати та керувати хмарними ресурсами, такими як віртуальні машини, сховища, мережі, бази даних та інше. Можна використовувати IaC для розгортання та керування інфраструктурою у будь-якому хмарному середовищі, такому як Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP) та інші.

Таким чином, інфраструктура як код є методологією та практикою, яка використовується для автоматизації управління інфраструктурними ресурсами, в тому числі у хмарних оточеннях.

Основні принципи та характеристики інфраструктури як код включають:

- декларативний підхід. IaC використовує декларативний підхід, де описується бажаний стан інфраструктури, а не конкретні кроки, необхідні для досягнення цього стану. Система самостійно приймає рішення про оптимальний спосіб досягнення цього стану;

- опис в коді полягає в тому, що інфраструктура (віртуальні машини, мережі, сховища, налаштування тощо) описується в коді за допомогою спеціальних мов програмування або шаблонів, які призначені для цього;

- автоматизація розгортання та повторюваність дозволяє знизити кількість помилок, забезпечити повторюваність та швидкість розгортання;

- відслідковуваність змін, а саме інфраструктура як код дозволяє відстежувати всі зміни та внесені правки до інфраструктури в систему контролю версій. Це забезпечує спрощене управління та відкат до попередніх станів системи;

- масштабованість. За допомогою IaC можна легко масштабувати інфраструктуру, збільшуючи або зменшуючи кількість ресурсів за потреби,

що сприяє ефективному використанню ресурсів;

- безпека. Використання IaC сприяє поліпшенню безпеки, оскільки можна встановлювати конфігураційні правила та забезпечувати дотримання стандартів безпеки;

- швидкість та ефективність полягає в тому, що IaC дозволяє швидко створювати та управляти інфраструктурою, знижувати час від розробки до впровадження, покращувати процеси розробки та забезпечувати високу продуктивність.

Інфраструктура як код використовується не лише у хмарних середовищах, але і в традиційних дата-центрах. Однак у хмарних середовищах цей підхід набуває особливої популярності, оскільки він дозволяє більш ефективно керувати та використовувати ресурси хмарних платформ. Переваги інфраструктури як код у хмарних оточеннях включають ефективніше керування ресурсами, швидше розгортання та забезпечення однорідності конфігурацій. Цей підхід сприяє забезпеченню стабільності, масштабованості та безпеки в управлінні хмарними послугами.

Таким чином, інфраструктура як код є логічним кроком для ефективного та автоматизованого управління хмарними ресурсами та оптимізації їх використання.

1.5 Засоби для реалізації інфраструктури як коду

Популярні засоби для реалізації інфраструктури як коду включають такі засоби та інструменти як Terraform, AWS CloudFormation, Ansible, Puppet, Chef та інші. Ці інструменти допомагають створювати, керувати та відстежувати інфраструктурні ресурси через програмний код [6].

До основних кроків розгортання інфраструктури належать:

- визначення вимог. Цей крок полягає в тому, що спочатку потрібно чітко виконати вимоги до інфраструктури. Це може включати обчислювальні ресурси, сховища даних, мережеві налаштування, безпеку тощо;

- вибір інструментів. Треба вибрати придатні інструменти та засоби для реалізації розгортання. Якщо використовується практика інфраструктури як коду, обирають відповідний інструмент;
- створення коду інфраструктури. Цей крок полягає в написанні коду, який описує всі необхідні ресурси та налаштування для інфраструктури. Цей код може бути структурованим за допомогою файлів або скриптів, залежно від обраного інструменту;
- тестування. Перед розгортанням необхідно провести тестування коду інфраструктури на відповідність вимогам та наявним стандартам;
- розгортання полягає у процесі самого розгортання, під час якого інструмент автоматично створює та налаштовує необхідні ресурси відповідно до коду;
- конфігурація та налаштування. Після розгортання можна додатково налаштувати ресурси, які вже створені;
- моніторинг та управління. Після розгортання слід встановити моніторинг для відстеження працездатності інфраструктури, а також забезпечити систему управління, щоб реагувати на зміни та події.

Важливо пам'ятати, що розгортання інфраструктури - це динамічний процес, який може змінюватися з часом під час зміни вимог до програмного забезпечення. IaC зберегти консистентність і контроль над інфраструктурою навіть під час різних змін.

1.6 Інструменти для розгортання інфраструктури як код на різних хмарних платформах

Розглянемо популярні інструменти для розгортання інфраструктури як код на різних хмарних платформах.

AWS CloudFormation - це сервіс управління інфраструктурою як коду, який надає можливість автоматизувати процес розгортання та управління ресурсами в AWS за допомогою шаблонів. AWS CloudFormation дозволяє

автоматизувати процес розгортання та управління інфраструктурою, що сприяє швидкому та ефективному створенню ресурсів. Шаблони AWS CloudFormation використовують декларативний підхід, де описується бажаний стан інфраструктури. Система сама вирішує, як досягти цього стану. Цей сервіс добре інтегрований з іншими послугами AWS, що спрощує розгортання складних архітектур та додаткових функцій. AWS CloudFormation забезпечує відновлення стабільного стану системи у випадку помилок чи відмов, забезпечуючи надійність. Система підтримує масштабування, що дозволяє ефективно керувати великими розгортаннями та динамічно змінювати їхню конфігурацію. Існує інтеграція з Terraform. Незважаючи на переваги, AWS CloudFormation має деякі недоліки:

- складність синтаксису, що полягає в тому, як шаблони CloudFormation можуть бути складними для створення та редагування, особливо для новачків. Синтаксичні помилки можуть важко виявлятися та виправлятися;

- обмеження та недоліки функціональності. Деякі досить специфічні або нові функції AWS можуть бути погано підтримувані в CloudFormation або взагалі відсутніми;

- відсутність можливості оновлення окремих ресурсів. При зміні конфігурації потрібно оновлювати всю стек-інфраструктуру, навіть якщо змінюється лише один ресурс;

- швидкість розгортання. Деякі користувачі відзначають, що швидкість розгортання інфраструктури в CloudFormation може бути помітно повільнішою порівняно з іншими інструментами.

Azure Resource Manager Templates є інструментом для визначення та управління інфраструктурою Azure як коду. Розглянемо переваги та недоліки цього підходу.

Переваги:

- ARM Templates інтегровані з Azure та повністю підтримують віртуальні машини, бази даних, сховища та інші ресурси Azure;

- шаблони використовують декларативний підхід, де описується бажаний стан інфраструктури, а не конкретні кроки для досягнення цього стану;

- ARM дозволяє легко масштабувати вашу інфраструктуру, створюючи декілька інстанцій ресурсів за допомогою одного шаблону;

- модульність та повторне використання також відноситься до переваг цього інструменту. Можна організувати шаблони в модулі для полегшення повторного використання та кращої організації коду;

- шаблони мають можливість версіонування, що спрощує управління та розгортаннями в різних середовищах;

- автоматизація розгортання та керування циклом життя полягає в тому, що ARM Templates дозволяють автоматизувати процес розгортання та керування циклом життя ресурсів, забезпечуючи ефективність та надійність.

Недоліки:

- складність синтаксису та вивчення;
- дефіцит вбудованих функцій;
- складність розробки та тестування;
- інтеграція з Terraform;
- залежність від екосистеми Azure.

Google Cloud Deployment Manager. Цей інструмент дозволяє описувати та розгортати інфраструктуру GCP за допомогою YAML-або Jinja-шаблонів.

Google Cloud Deployment Manager є інструментом для автоматизації та управління інфраструктурою як коду в Google Cloud Platform (GCP). Розглянемо його переваги та недоліки.

Google Cloud Deployment Manager інтегрований із сервісами та ресурсами GCP, дозволяючи ефективно керувати інфраструктурою GCP. Шаблони Deployment Manager також використовують декларативний підхід, де описується бажаний стан інфраструктури, а не конкретні кроки для досягнення цього стану. Deployment Manager дозволяє організувати шаблони в модулі для полегшення повторного використання та кращої організації

коду. Шаблони мають можливість версіонування, що спрощує управління та розгортаннями в різних середовищах. Deployment Manager легко інтегрується з іншими сервісами GCP, що дозволяє ефективно керувати інфраструктурою та додавати додатковий функціонал.

Але також є недоліки. У порівнянні з AWS CloudFormation або Azure Resource Manager, Deployment Manager може мати меншу кількість готових або офіційно підтримуваних шаблонів. Синтаксис шаблонів може бути складним для новачків та вимагати додаткового часу для вивчення. Deployment Manager може мати меншу спільноту користувачів порівняно з іншими інструментами IaC, що може вплинути на доступність допомоги та розроблених рішень. Deployment Manager призначений для використання виключно з GCP, тому не може бути використаний для керування інфраструктурою на інших хмарних платформах.

IBM Cloud Schematics. Цей сервіс дозволяє автоматизувати розгортання та управління ресурсами на IBM Cloud за допомогою Terraform, Ansible та інших інструментів. Переваги:

- інтеграція з IBM Cloud;
- декларативний підхід;
- модульність та повторне використання;
- автоматизація та масштабованість;
- інтеграція з Terraform.

Недоліки:

- має меншу спільноту користувачів порівняно з більш популярними IaC інструментами;
- менша кількість готових рішень;
- відсутність підтримки інших хмарних платформ;
- можливі обмеження функціональності. Деякі специфічні або нові функції IBM Cloud можуть бути погано підтримувані в IBM Cloud Schematics або взагалі відсутніми.

Oracle Cloud Infrastructure (OCI) Resource Manager - це сервіс

управління інфраструктурою як коду в Oracle Cloud за допомогою Terraform-конфігурацій. Основні переваги цього інструменту:

- інтеграція з Oracle Cloud;
- декларативний підхід;
- модульність та повторне використання;
- автоматизація та масштабованість;
- інтеграція з Terraform. Resource Manager від Oracle Cloud підтримує Terraform, що розширює можливості використання IaC у середовищі OCI.

До основних недоліків можна віднести обмежену спільноту, порівняно з більш популярними IaC інструментами, що може вплинути на доступність допомоги та розроблених рішень, обмежена функціональність, відсутність підтримки інших хмарних платформ.

Таким чином, при порівнянні стандартних інструментів для розгортання інфраструктури як коду в різних хмарних платформах: AWS CloudFormation, Azure Resource Manager Templates, Google Cloud Deployment Manager, IBM Cloud Schematics та Oracle Cloud Infrastructure Resource Manager можна сказати про переваги загалом:

- інтеграція та декларативний підхід. Всі платформи використовують декларативний підхід та глибоку інтеграцію з відповідними хмарними середовищами, та з Terraform;
- модульність та повторне використання. Всі інструменти дозволяють організувати конфігурації в модулі для кращої організації та повторного використання коду;
- автоматизація та масштабованість. Всі платформи дозволяють автоматизувати процеси розгортання та масштабування інфраструктури.

Також можна сказати і про недоліки загалом:

- обмежена спільнота. Деякі інструменти можуть мати меншу спільноту користувачів, що може вплинути на доступність допомоги та розроблених рішень;
- відсутність підтримки інших платформ. Кожен з інструментів

призначений для використання відповідно в різних хмарних середовищах та не має повноцінної підтримки інших платформ.

Висновок щодо стандартних популярних інструментів для розгортання інфраструктури як код на різних хмарних платформах наведений у таблиці (таблиця 1.1.).

Таблиця 1.1 – Основні переваги стандартних інструментів для розгортання інфраструктури як код на різних хмарних платформах

Інструмент для розгортання IaC	Основні переваги
AWS CloudFormation	Сильна інтеграція з AWS та широкий функціонал. Основний вибір для AWS
ARM Templates	Глибока інтеграція з Azure, має багато можливостей. Основний вибір для Azure
Google Cloud Deployment Manager	Ефективний для розгортання та управління інфраструктурою в Google Cloud
IBM Cloud Schematics	Для тих, хто в основному використовує IBM Cloud. Має деякі обмеження порівняно з іншими
Oracle Cloud Infrastructure Resource Manager	Для користувачів Oracle Cloud. Глибока інтеграція з Oracle Cloud Infrastructure

Треба зазначити, що кожен з інструментів для розгортання інфраструктури як код на різних хмарних платформах підтримує інтеграцію з Terraform.

Terraform є одним з найпопулярніших інструментів для оркестрації та управління інфраструктурою як код. Його популярність визначається кількома ключовими факторами:

- мультиплатформність. Terraform підтримує багато хмарних платформ, провайдерів та середовищ (AWS, Azure, Google Cloud, Oracle

Cloud, і багато інших). Це робить його універсальним інструментом для оркестрації різноманітних інфраструктур;

- декларативний підхід. Terraform використовує декларативний підхід до опису інфраструктури, де ви описуєте бажаний стан системи у конфігураційних файлах. Система сама вирішує, як досягти цього стану, спрощуючи процес розгортання та управління;

- простота використання. Terraform має легкий до зрозуміння синтаксис, оснований на HCL (HashiCorp Configuration Language), який дозволяє швидко створювати та редагувати конфігураційні файли;

- масштабованість та модульність. Terraform дозволяє масштабувати конфігурації та використовувати модулі для організації та повторного використання коду. Це особливо важливо для великих проектів зі складною інфраструктурою;

- інтеграція з іншими інструментами. Terraform легко інтегрується з іншими інструментами та системами контролю версій, дозволяючи автоматизувати процеси розгортання та управління інфраструктурою;

- спільнота та підтримка. Terraform має активну спільноту користувачів, яка сприяє обміну досвідом та наданню порад та рекомендацій.

Основні переваги Terraform полягають у його потужних можливостях, швидкості та простоті використання, що робить його ідеальним вибором для багатьох проектів та організацій, що працюють з хмарними та інфраструктурними ресурсами.

Незважаючи на свою популярність та ефективність, Terraform також має свої недоліки та виклики, які важливо враховувати, а саме:

- складність для великих проектів. У великих та складних проектах керування та утримання великої кількості конфігурацій може стати викликом. Код може стати набагато складнішим та важчим для розуміння;

- нестабільність певних функцій. У нових версіях Terraform можуть виникати зміни та покращення, але це також може спричинити проблеми сумісності з попередніми версіями та викликати непередбачувану поведінку;

- схильність до помилок конфігурації. Інструменти IaC, включаючи Terraform, можуть вносити залежність від конкретного провайдера хмарних послуг, що ускладнює переносимість конфігурацій між платформами. Некоректно сконфігурований код Terraform може призвести до непередбачуваних результатів та проблем у роботі інфраструктури;

- відсутність вбудованого циклу життя ресурсів. Terraform не має вбудованого механізму для повного циклу життя ресурсів (створення, зміну, видалення). Деякі операції можуть вимагати додаткових скриптів та інструментів.

Для успішного використання Terraform важливо ретельно планувати та управляти процесом розгортання та управління інфраструктурою, а також слідкувати за оновленнями та змінами у нових версіях.

1.7 Постановка задачі дослідження

Метою кваліфікаційної роботи є дослідження методів та засобів автоматичного розгортання інфраструктури як коду для хмарних сервісів.

Для дослідження методів та засобів автоматичного розгортання інфраструктури як коду для хмарних сервісів були обрані популярні інструменти автоматичного розгортання інфраструктури як коду провідних провайдерів хмарних платформ, а саме: AWS CloudFormation, Azure Resource Manager Templates, Google Cloud Deployment Manager, IBM Cloud Schematics та Oracle Cloud Infrastructure Resource Manager. В якості основного хмарного провайдера був обраний Microsoft Azure. Для практичної реалізації завдання автоматичного розгортання інфраструктури як коду були обрані засоби, що мають найбільшу кількість функціональних можливостей IaC такі як Azure Resource Manager Templates, доменно-орієнтована мова Вісер та мультиплатформний Terraform.

2 МЕТОДОЛОГІЯ АВТОМАТИЧНОГО РОЗГОРТАННЯ ІНФРАСТРУКТУРИ ЯК КОДУ ДЛЯ ХМАРНИХ СЕРВІСІВ

2.1 Основні кроки розгортання інфраструктури

Популярні рішення для реалізації інфраструктури як коду включають такі засоби та інструменти як Terraform, AWS CloudFormation, Azure Resource Manager Templates та інші. Ці інструменти допомагають створювати, керувати та відстежувати інфраструктурні ресурси через програмний код [6].

До основних кроків розгортання інфраструктури належать:

- визначення вимог. Спочатку потрібно чітко виконати вимоги до інфраструктури. Це може включати обчислювальні ресурси, сховища даних, мережеві налаштування, безпеку тощо;

- вибір інструментів. Треба вибрати придатні інструменти та засоби для реалізації розгортання. Якщо використовується практика інфраструктури як коду, обирають відповідний інструмент, наприклад, Terraform, CloudFormation, ARM Templates тощо;

- створення коду інфраструктури. Створюють код, який описує всі необхідні ресурси та налаштування для інфраструктури. Цей код може бути структурованим за допомогою файлів або скриптів, залежно від обраного інструменту;

- тестування. Перед розгортанням проводять тестування коду інфраструктури на відповідність вимогам та наявним стандартам;

- розгортання. Запуск процесу розгортання, під час якого інструмент автоматично створює та налаштовує необхідні ресурси відповідно до коду;

- конфігурація та налаштування. Після розгортання можна додатково налаштувати ресурси, які вже створені;

- моніторинг та управління. Після розгортання слід встановити

моніторинг для відстеження працездатності інфраструктури, а також забезпечити систему управління, щоб реагувати на зміни та події.

Важливо пам'ятати, що розгортання інфраструктури - це динамічний процес, який може змінюватися з часом під час зміни вимог до програмного забезпечення. IaC зберегти консистентність і контроль над інфраструктурою навіть під час різних змін.

2.2 Етапи класичного розгортання інфраструктури

Традиційне розгортання, класичне, означає налаштування та встановлення інфраструктурних ресурсів, таких як сервери, мережеві компоненти, бази даних тощо, вручну або за допомогою засобів, які не є автоматизованими. Однак цей підхід стає менш популярним при застосуванні автоматизованого розгортання за допомогою практичної інфраструктури як коду. Основні етапи класичного розгортання інфраструктури:

- по-перше, це планування, тобто визначення вимог до інфраструктури, вибір обладнання, обчислювальних ресурсів, мережевої архітектури тощо;
- по-друге - придбання фізичного обладнання або оренда серверів у дата-центрі;
- по-третє - розміщення серверів, налаштування мережі, підключення до джерел електроживлення, охолодження тощо;
- потім встановлення операційної системи на кожному сервері, налаштування мережі, безпеки та інше. А також встановлення та конфігурація сховищ даних (наприклад, база даних), які використовуються додатками;
- наступний етап - це налаштування заходів безпеки, включаючи файрволі, антивіруси, моніторинг та інші заходи;
- далі - безпосередньо розгортання програмного забезпечення: встановлення та конфігурація програмного забезпечення, яке буде працювати

на інфраструктурі;

- передостанній етап полягає у тестування, виконанні тестів для перевірки працездатності та стабільності системи;

- і, на останок, додаткове управління та підтримка інфраструктури, включаючи регулярне оновлення, моніторинг та відповідь на проблеми.

Класичний підхід до розгортання інфраструктури може бути працездатним, але він може бути гнучким, вимагати більше часу на ручне налаштування та зміну інфраструктури. У сучасному світі більше підприємств переходять до автоматизованих практик IaC для забезпечення більшої ефективності та узгодженості в розгортанні та керуванні інфраструктурою.

2.3 Основні автоматизовані практики IaC

Автоматизовані практики інфраструктури як коду - це підхід до розгортання та управління інфраструктурою, коли всі дії, пов'язані зі створенням, налаштуваннями та управлінням ресурсами, забезпечуються за допомогою програмного коду. Основною метою є автоматизація процесів, забезпечення сумісності, зменшення ризиків помилок та забезпечення гнучкості в управлінській інфраструктурі.

Ключові практики IaC включають:

- декларативний код. Код IaC описує бажану структуру та налаштування інфраструктури, а не послідовність дій для його створення. Це дозволяє системі самостійно розпізнавати, які зміни потрібно зробити, для досягнення бажаного стану;

- код IaC може бути збережений та відстежуваний у системі керування версіями, що дозволяє відновлювати попередні стани та вести спільну роботу в команді;

- за допомогою IaC можна легко розгортати та масштабувати ресурси відповідно до потреб;

- код IaC може бути застосований для розгортання інфраструктури на різних етапах розробки та в різних середовищах (наприклад, розробка, тестування, продакшн);

- автоматизована та неперервна доставка. IaC допоможе забезпечити автоматичне розгортання та оновлення інфраструктури, що підтримує практику неперервної доставки;

- тести та перевірки. Код IaC може бути підданим автоматизованим тестуванням, щоб виявити помилки до розгортання.

Використання автоматизованих практик IaC дозволить зберегти час, зменшити ризики, підвищити сумісність та спростити процеси розгортання та управління інфраструктурою.

2.4 Доменно-орієнтована мова для опису та розгортання інфраструктури

При розгортанні IaC все частіше використовують доменно-орієнтовану мову (Domain-Specific Language, DSL) - це спеціалізована мова програмування або конфігурації, яка розроблена для виразного опису інфраструктури та її розгортання. Основна ідея перетворюється в тому, щоб мову, спеціально налаштовану для виразу концепцій і понять, що відповідає конкретному домену (у цьому випадку - розгортанню інфраструктури), замість використання загальної мови програмування.

Відмінність DSL від загальних мов програмування, таких як Python, JavaScript чи Ruby, виникає в тому, що DSL спрощує виразність, зрозумілість та специфічність для конкретної області, в цьому випадку - управління інфраструктурою. Це робить код більш читабельним і зрозумілим для тих, хто працює в даній області.

Кілька прикладів DSL для розгортання Інфраструктури як коду:

- Terraform. Терміни та конструкція Terraform - це приклад DSL. Файли конфігурації Terraform містять декларативний код, який описує

ресурси, їх взаємозв'язки та налаштування;

- CloudFormation. Аналогічно, шаблони CloudFormation для AWS також є формою DSL, спеціалізованою на описі ресурсів та послуг Amazon Web Services. Використання DSL робить роботу з розгортанням IaC більш зрозумілою та ефективною, особливо для тих, хто спеціалізується на даній області, а не на загальному програмуванні;

- у Microsoft Azure використання доменно-орієнтованої мови (DSL) дозволяє вам описати і керувати різними аспектами інфраструктури та ресурсів, які розгортаються в хмарному середовищі Azure [6]. Основним інструментом для використання DSL в Azure є мова ARM (Azure Resource Manager). Шаблони Azure Resource Manager - це JSON-подібна мова, яка дозволяє створювати ARM-шаблони - описи файлів, які використовують ресурси, їх взаємозв'язки та налаштування. Це DSL для розгортання та управління ресурсами Azure. У шаблонах ARM ви можете описати різні ресурси - від віртуальних машин та мережевих складових до баз даних та інших послуг. Azure CLI - це командний рядок Azure, який також має підтримку для використання DSL. Можна використовувати команди CLI для створення, налаштування та керування ресурсами за допомогою команд спеціалізованої мови.

3 ВИБІР МЕТОДІВ ТА ЗАСОБІВ ДЛЯ ПРАКТИЧНОЇ РЕАЛІЗАЦІЇ ЗАВДАННЯ АВТОМАТИЧНОГО РОЗГОРТАННЯ ІНФРАСТРУКТУРИ ЯК КОД

На основі аналізу, проведеного в попередніх розділах, можна зробити висновок, що Azure швидко розвивається та постійно вдосконалюється, особливо в контексті управління життєвим циклом розробки та аналітики даних. Azure пропонує деякі унікальні послуги та інструменти, які можуть бути важливими для деяких проектів. Для організацій, які вже використовують продукти Microsoft, Azure може бути більш зручним вибором, оскільки має сильну інтеграцію з іншими продуктами та технологіями Microsoft. Azure має розвинену мережу центрів даних у різних країнах світу, що може бути важливим фактором для організацій з географічно розподіленою інфраструктурою. В залежності від конкретних потреб, моделі ліцензування та ціноутворення Azure можуть бути більш зрозумілими або вигідними для певних типів користувачів.

Якщо обговорювати автоматичне розгортання інфраструктури, то ARM Templates охоплюють широкий спектр ресурсів та послуг Azure, включаючи віртуальні машини, бази даних, сховища, мережі та інші. Це дозволяє повністю керувати різними аспектами інфраструктури. ARM Templates мають значну історію використання та розвитку, що вказує на стабільність та довготривалу підтримку. Спільнота користувачів ARM постійно зростає, що обіцяє швидке вирішення питань та доступ до широкого спектру знань.

ARM Templates і Bicep - це два підходи до декларативного опису інфраструктури в Microsoft Azure. Ці підходи допомагають автоматизувати процес розгортання та управління ресурсами в хмарному середовищі. Bicep - це мова декларативного опису інфраструктури, розроблена спеціально для Microsoft Azure. Вона є альтернативною мовою ARM Templates, яка запускає процес опису та управління ресурсами в хмарному середовищі Azure. Bicep

— це доменно-орієнтована мова (DSL), яка використовує декларативний синтаксис для розгортання ресурсів Azure. У файлі Bicep визначається інфраструктура, яку хочете розгорнути в Azure, а потім використовуєте цей файл протягом усього життєвого циклу розробки для повторного розгортання інфраструктури. Для використання DSL в Azure потрібно ознайомитися з документацією, прикладами та практичними завданнями для кожного інструмента або підходу. Кожен інструмент має свої особливості, але загальна ідея виникає в тому, щоб описати потрібний стан інфраструктури у вигляді коду та виконати його для розгортання та керування ресурсами Azure. Для розгортання інфраструктури в Azure можна також використовувати і Terraform. ARM Templates, Bicep та Terraform - це популярні інструменти для розгортання та керування інфраструктурою як коду в Microsoft Azure [6]. Розглянемо їх у порівнянні.

ARM Templates є стандартними для роботи з Azure і мають велику кількість ресурсів та функціональності. Azure Resource Manager Templates використовують мову JSON для опису ресурсів та їх взаємозв'язків. JSON може бути важким для читання та редагування, особливо для складних конфігурацій.

ARM Templates можуть стати досить великими і складними через велику кількість JSON-коду, що розбавляє структурованість та читабельність.

Bicep використовує більш декларативну мову, яка є спеціальною для Azure і розроблена для зрозумілості та структурованості. Bicep набагато читабельніший, оскільки має більш лаконічну синтаксичну структуру та менше пунктуації. Bicep дозволяє створювати більш компактні та зрозумілі описи інфраструктури, завдяки чому ваш код залишається більш структурованим. Bicep має підтримку модулів, які сприяють перевикористанню коду та полегшують підтримку.

Основна ідея Bicep полягає в тому, щоб зробити опис інфраструктури більш зрозумілим, менш складним і більш структурованим. Bicep дозволяє

більш ефективно та зрозуміло працювати з розгортанням інфраструктури в Azure, а також полегшує роботу з ARM Templates.

ARM Templates і Bicep - це засоби для декларативного опису та розгортання інфраструктури в Microsoft Azure [6, 10]. Порівняння двох цих засобів наведено у таблиці (таблиця 3.1.).

Таблиця 3.1 – Порівняння засобів декларативного опису та розгортання інфраструктури в Microsoft Azure

Критерії порівняння до засобів	ARM Templates	Bicep
1	2	3
Мова	ARM Templates вибір JSON для опису ресурсів та їх налаштування. JSON є достатнім стандартним форматом, але великі шаблони можуть бути складними для читання та редагування.	Bicep використовує більш декларативний та структурований синтаксис, що надає властивості мови програмування. Це полегшує розуміння та редагування.
Зрозумілість	Доволі часто шаблони ARM можуть стати заплутаними через велику кількість JSON-коду та потребують вказати досить багато деталей.	Bicep робить код більш зрозумілим за допомогою свого синтаксису. Це особливо важливо при роботі з великими та складними шаблонами.
постережуваність	налагоджено відстежити зміни та розрізнити, які	стають більш видимими, щоб ви могли переглядати

Продовження таблиці 3.1

1	2	3
	ресурси змінені чи додані.	сирцевий Вісер-код, який згенерував певний шаблон ARM.
Модульність	Підтримка модульності та перевикористання коду не завжди є зручною, особливо для складних шаблонів.	Вісер має кращу підтримку модульності та перевикористання коду. Можна створювати власні модулі та використовувати їх для спрощення шаблонів.

Вісер дозволяє більш ефективно та зрозуміло працювати з розгортанням інфраструктури в Azure, а також полегшує роботу з ARM Templates. Загалом, Вісер може бути більш зручним та зрозумілим у порівнянні з шаблонами ARM, особливо для тих, хто має досвід програмування мовами. Вісер може бути кращим вибором для тих, хто шукає більш лаконічний та читабельний спосіб опису інфраструктури в Azure.

Основна перевага Вісер полягає в тому, щоб полегшити створення та управління інфраструктурою в Azure, зробивши код більш зрозумілим та ефективним. Файли Вісер компілюються у шаблони ARM, тож вони можуть бути легко використані та розгорнуті з використанням ARM [7].

Terraform підтримує багато хмарних платформ, дозволяючи управляти інфраструктурою в AWS, Azure, Google Cloud та інших середовищах. Terraform має велику кількість провайдерів, що надає можливість керувати різними типами ресурсів у різних платформах. Його конфігурації можна створювати у простому та легкому для сприйняття синтаксисі HCL (HashiCorp Configuration Language). Також Terraform має велику активну спільноту та багату екосистему модулів.

Наприкінці можна зробити наступний висновок.

Bicep відрізняється простим та зрозумілим синтаксисом, ARM має глибоку інтеграцію з Azure, а Terraform є платформно-незалежним (мультиплатформним) та має простий синтаксис. Bicep та Terraform підтримують модульність та повторне використання, що сприяє ефективнішому управлінню конфігураціями. Terraform має перевагу у багатьох провайдерах та широкому спектрі можливостей для управління інфраструктурою в різних хмарних платформах.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

В попередніх розділах були досліджені методи та засоби автоматичного розгортання інфраструктури як коду для хмарних сервісів провідних хмарних платформ. Для практичної реалізації завдання автоматичного розгортання інфраструктури як коду були обрані засоби, що мають найбільшу кількість функціональних можливостей IaC такі як Azure Resource Manager Templates, доменно-орієнтована мова Bicep та мультиплатформний Terraform.

4.1 Автоматичне розгортання інфраструктури як коду за допомогою Azure Resource Manager Templates

Azure Resource Manager Templates (ARM Templates) - це декларативний спосіб опису інфраструктури та сервісів Azure, який був запущен у 2014 році як частина Azure Resource Manager, інструменту управління ресурсами від Microsoft Azure. Використання ARM Templates дозволяє інженерам визначати, налаштовувати, та управляти ресурсами Azure за допомогою JSON-файлів, що забезпечує зручність та однорідність конфігурації.

У ARM Templates немає загальної версії для всього шаблону. Замість цього, кожен ресурс використовує свою версію API, яка вказується в конфігурації ресурсу в шаблоні. Ці версії API визначають доступні властивості та функції для кожного типу ресурсу і оновлюються Microsoft незалежно.

Основні переваги ARM Templates включають:

- ідемпотентність. Кожен раз, коли шаблон застосовується, він гарантує, що ресурси будуть розгорнуті в точно такому ж стані, незалежно від поточного стану середовища;
- модульність. Шаблони можна розділити на менші модулі, які можуть

бути повторно використані в різних середовищах та проекта;

- управління залежностями. ARM Templates автоматично розпізнають та управляють залежностями між ресурсами, забезпечуючи правильний порядок їх створення.

Процес розгортання з ARM Templates можна описати наступними етапами:

- підготовка шаблону. Розробка шаблону ARM починається з визначення ресурсів, які потрібно розгорнути. Це можуть бути віртуальні машини, мережеві компоненти, бази даних, а також багато інших типів ресурсів, доступних в Azure;

- параметризація. Для забезпечення гнучкості та можливості повторного використання шаблонів, визначення ресурсів зазвичай параметризуються. Це дозволяє користувачам вносити специфічні значення під час кожного розгортання, наприклад, розмір віртуальної машини або ім'я бази даних, не змінюючи основний шаблон;

- валідація шаблону. Перед розгортанням шаблону важливо переконатися, що він не містить синтаксичних помилок і що всі вказані ресурси можуть бути створені. Azure надає інструменти для валідації шаблонів, що допомагають уникнути помилок під час розгортання;

- розгортання шаблону. Після валідації шаблону його можна розгорнути в Azure. Це можна зробити через портал Azure, використовуючи Azure PowerShell, Azure CLI, або через системи неперервної інтеграції/неперервного розгортання (CI/CD);

- управління станом. ARM Templates дозволяють не тільки створювати, але й управляти станом ресурсів. Якщо в шаблоні внести зміни та повторно його розгорнути, Azure Resource Manager здійснить зміни в існуючих ресурсах, щоб відповідати новому стану, описаному в шаблоні;

- моніторинг та діагностика. Після розгортання ARM Templates активно використовують інструменти моніторингу та діагностики, щоб забезпечити візуалізацію стану ресурсів та швидке виявлення та реагування

на проблеми. Azure Monitor та Azure Service Health пропонують комплексні можливості для відстеження стану ресурсів, аналізу логів та отримання сповіщень про критичні події;

- версіонування та управління змінами. Щоб впоратися зі змінами в інфраструктурі протягом часу, важливо вести версіонування шаблонів ARM. Застосування практик управління версіями, таких як зберігання шаблонів у системах контролю версій (наприклад, Git), дозволяє з легкістю відстежувати зміни та управляти різними версіями інфраструктури. Це також сприяє кращому співробітництву між членами команди та забезпечує можливість відкату до попередніх версій, якщо виникає потреба;

- безпека та дотримання стандартів. Під час розробки шаблонів ARM важливо включати налаштування безпеки та контролю доступу. Azure Policy та ролі доступу на основі Azure Role-Based Access Control (RBAC) забезпечують інструменти для автоматизації управління політиками та дотриманням стандартів безпеки.

Для ілюстрації вищезазначених принципів розглянемо конкретний сценарій розгортання веб-додатку в Azure за допомогою ARM шаблону. Шаблон включає в себе створення ресурсної групи (resource group), веб-додатку (Web App), бази даних SQL Server, Storage Account та Key Vault. Важливою особливістю є параметризація, яка дозволяє користувачу визначати імена, розміри та інші конфігураційні параметри для кожного ресурсу під час розгортання. Також важливо додати, що вигадана інфраструктура захищена Azure Policy.

Валідація шаблону перед розгортанням гарантує, що всі ресурси правильно сконфігуровані та готові до створення. Після успішного розгортання, шаблон автоматично згенерує connection string для Storage Account та надасть можливість його збереження в створений Key Vault, забезпечуючи безпеку та централізоване управління конфіденційною інформацією.

Такий підхід не лише спрощує процес розгортання та управління

інфраструктурою, але й знижує ймовірність помилок, які можуть виникнути при ручному налаштуванні.

Лістинг 4.1 – Код розгортання наведеної інфраструктури за допомогою Azure Resource Manager Templates

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-
04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string",
      "defaultValue": "East US"
    },
    "webAppName": {
      "type": "string"
    },
    "sqlServerName": {
      "type": "string"
    },
    "sqlDatabaseName": {
      "type": "string"
    },
    "storageAccountName": {
      "type": "string"
    },
    "keyVaultName": {
      "type": "string"
    }
  },
  "variables": {},
  "resources": [
    {
      "type": "Microsoft.Resources/resourceGroups",
      "apiVersion": "2021-04-01",
      "name": "example-resources",
      "location": "[parameters('location')]"
    },
    {
      "type": "Microsoft.Web/serverfarms",
      "apiVersion": "2021-01-15",
      "name": "[concat(parameters('webAppName'), '-service-
plan')]",
      "location": "[parameters('location')]",
      "properties": {
        "reserved": false
      },
```

```
      "sku": {
        "name": "S1",
        "tier": "Standard"
      }
    },
    {
      "type": "Microsoft.Web/sites",
      "apiVersion": "2021-01-15",
      "name": "[parameters('webAppName')]",
      "location": "[parameters('location')]",
      "properties": {
        "serverFarmId":
"[resourceId('Microsoft.Web/serverfarms',
concat(parameters('webAppName'), '-service-plan'))]"
      }
    },
    {
      "type": "Microsoft.Sql/servers",
      "apiVersion": "2020-11-01-preview",
      "name": "[parameters('sqlServerName')]",
      "location": "[parameters('location')]",
      "properties": {
        "administratorLogin": "sqladmin",
        "administratorLoginPassword": "Pa$$w0rd123",
        "version": "12.0"
      }
    },
    {
      "type": "Microsoft.Sql/servers/databases",
      "apiVersion": "2020-11-01-preview",
      "name": "[concat(parameters('sqlServerName'), '/',
parameters('sqlDatabaseName'))]",
      "properties": {
        "collation": "SQL_Latin1_General_CP1_CI_AS",
        "edition": "Basic"
      },
      "dependsOn": [
        "[resourceId('Microsoft.Sql/servers',
parameters('sqlServerName'))]"
      ]
    },
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2021-06-01",
      "name": "[parameters('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "StorageV2"
    },
    {
      "type": "Microsoft.KeyVault/vaults",
```

```
      "apiVersion": "2021-06-01-preview",
      "name": "[parameters('keyVaultName')]",
      "location": "[parameters('location')]",
      "properties": {
        "sku": {
          "family": "A",
          "name": "standard"
        },
        "tenantId": "[subscription().tenantId]"
      }
    },
    {
      "type": "Microsoft.KeyVault/vaults/secrets",
      "apiVersion": "2021-06-01-preview",
      "name": "[concat(parameters('keyVaultName'),
'/storageAccountKey')]",
      "properties": {
        "value":
"[listKeys(resourceId('Microsoft.Storage/storageAccounts',
parameters('storageAccountName')), '2021-06-01').keys[0].value]"
      },
      "dependsOn": [
        "[resourceId('Microsoft.KeyVault/vaults',
parameters('keyVaultName'))]",
        "[resourceId('Microsoft.Storage/storageAccounts',
parameters('storageAccountName'))]"
      ]
    }
  ]
}
```

4.2 Автоматичне розгортання інфраструктури як коду за допомогою Bicep

Bicep, розроблений Microsoft і вперше представлений у 2020 році, є новітнім інструментом для декларативного опису інфраструктури в Azure. Цей інструмент використовує спрощений синтаксис у порівнянні з традиційними ARM Templates, забезпечуючи розробникам більш зручний і чистий спосіб опису інфраструктури через код. Створений як відповідь на потребу в простоті та зрозумілості, Bicep підвищує читабельність і спрощує складність конфігурацій, роблячи процес створення та управління ресурсами Azure більш інтуїтивно зрозумілим.

Важливо відзначити, що в процессі розгортання Bicep автоматично

перетворює свій код на ARM шаблони, що гарантує плавне та безперебійне інтегрування з уже існуючою інфраструктурою Azure. Ця функція робить Вісер ефективним інструментом у процесі розробки та управління ресурсами хмарних сервісів.

У Вісер, подібно до ARM Templates, не існує єдиної загальної версії для всього шаблону. Замість цього, в Вісер кожен ресурс використовує свою окрему версію API, яка вказується у визначенні кожного ресурсу. Ці версії API визначають набір доступних властивостей та функціональностей для кожного типу ресурсу Azure і оновлюються Microsoft незалежно. Такий підхід дозволяє використовувати останні можливості, які надає Azure, одночасно забезпечуючи гнучкість у роботі з різними версіями API.

Основні переваги Вісер включають:

- читабельність та лаконічність: код Вісер легший для читання та написання, ніж відповідний JSON код ARM шаблонів, завдяки його чіткій та концисній структурі;

- підтримка модульності: Вісер дозволяє розбивати конфігурацію на модулі, що полегшує повторне використання коду і організацію проектів;

- вбудоване управління залежностями: Вісер автоматично управляє залежностями між ресурсами, забезпечуючи ефективне створення і конфігурацію інфраструктури.

Процес розгортання з Вісер включає:

- розробка шаблону: Написання коду Вісер для визначення необхідних ресурсів, таких як веб-додатки, бази даних, Storage Accounts та інші компоненти Azure;

- параметризація: використання параметрів для надання гнучкості та налаштування різних аспектів інфраструктури, наприклад, вказівка розміру інстансів або назв ресурсів;

- валідація та компіляція: перевірка Вісер коду на помилки перед компіляцією його в ARM шаблон JSON формату, який може бути розгорнутий в Azure;

- розгортання: застосування скомпільованого шаблону в Azure за допомогою Azure CLI, PowerShell або через CI/CD процеси для створення та конфігурації інфраструктури;

- інтеграція з Azure Services: використання інструментів, таких як Azure Policy та Azure Role-Based Access Control, для забезпечення безпеки та відповідності стандартам.

Розглянемо той же самий сценарій, що й для ARM Templates, де Bicep використовується для створення веб-додатку, бази даних SQL Server, Storage Account та Key Vault в Azure. Шаблон Bicep забезпечує не тільки створення цих ресурсів, але й генерацію connection string для Storage Account та його безпечне зберігання у Key Vault. Це дозволяє автоматизувати та спростити управління конфіденційними даними, а також забезпечує централізоване управління доступом до ресурсів.

Застосування Bicep в цьому контексті сприяє чіткій організації коду, зменшує ймовірність помилок при ручному налаштуванні інфраструктури та підвищує ефективність управління хмарними ресурсами.

Лістинг 4.2 – Код розгортання наведеної інфраструктури за допомогою Bicep

```
param location string = 'East US'
param webAppName string
param sqlServerName string
param sqlDatabaseName string
param storageAccountName string
param keyVaultName string

resource resourceGroup 'Microsoft.Resources/resourceGroups@2021-
04-01' = {
  name: 'example-resources'
  location: location
}

resource appServicePlan 'Microsoft.Web/serverfarms@2021-01-15' =
{
  name: '${webAppName}-service-plan'
  location: location
  properties: {
```

```
    reserved: false
  }
  sku: {
    name: 'S1'
    tier: 'Standard'
  }
}

resource webApp 'Microsoft.Web/sites@2021-01-15' = {
  name: webAppName
  location: location
  properties: {
    serverFarmId: appServicePlan.id
  }
}

resource sqlServer 'Microsoft.Sql/servers@2020-11-01-preview' =
{
  name: sqlServerName
  location: location
  properties: {
    administratorLogin: 'sqladmin'
    administratorLoginPassword: 'Pa$$w0rd123'
    version: '12.0'
  }
}

resource sqlDatabase 'Microsoft.Sql/servers/databases@2020-11-
01-preview' = {
  name: sqlDatabaseName
  properties: {
    collation: 'SQL_Latin1_General_CP1_CI_AS'
    edition: 'Basic'
  }
  dependsOn: [
    sqlServer
  ]
}

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-
06-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}

resource keyVault 'Microsoft.KeyVault/vaults@2021-06-01-preview'
= {
  name: keyVaultName
  location: location
```

```
  properties: {
    sku: {
      family: 'A'
      name: 'standard'
    }
    tenantId: subscription().tenantId
  }
}


resource keyVaultSecret 'Microsoft.KeyVault/vaults/secrets@2021-
06-01-preview' = {
  name: '${keyVaultName}/storageAccountKey'
  properties: {
    value: storageAccount.listKeys().keys[0].value
  }
  dependsOn: [
    keyVault
    storageAccount
  ]
}
```

4.3 Автоматичне розгортання інфраструктури як коду за допомогою Terraform

Terraform — це потужний інструмент для управління інфраструктурою як кодом, розроблений HashiCorp у 2014 році. Він дозволяє розробникам використовувати декларативний синтаксис для опису інфраструктури в коді, який потім може бути розгорнутий в різних хмарних провайдерах, включаючи Azure, AWS, Google Cloud Platform та інших.

У Terraform, розробленого HashiCorp, немає єдиної версії для всієї конфігурації або шаблону, але сам інструмент Terraform, як програмне забезпечення, має свої версії. Ці версії Terraform відображають зміни та оновлення в його функціональності, включаючи покращення в роботі з провайдерами, нові функції, виправлення помилок та інші оновлення. Кожна нова версія Terraform може вносити зміни, які впливають на спосіб управління інфраструктурою або на сумісність з різними провайдерами та їхніми версіями API. Провайдери Terraform, які є окремими компонентами для взаємодії з різними хмарними платформами, також мають свої версії та оновлюються незалежно, що дозволяє використовувати останні можливості

кожної хмарної платформи. Конфігурації Terraform визначають ресурси та їх параметри, але самі по собі не мають версій, на відміну від ресурсів у ARM Templates, де кожен ресурс має вказану версію API.

Провайдер Azure для Terraform, розроблений HashiCorp, розпочав свій шлях у 2014 році одночасно з випуском самого Terraform. З того часу, він постійно розвивався, регулярно додаючи підтримку нових ресурсів та функцій Azure. Станом на 19 січня 2024 року, остання версія провайдера Azure для Terraform – це 3.88.0, випщуена 17 січня 2024 року. Провайдер Azure регулярно оновлюється, приблизно кожні 10 днів, забезпечуючи користувачам актуальні функції та вдосконалення для ефективного управління інфраструктурою Azure.

Основні переваги Terraform включають:

- універсальність: Terraform підтримує численні хмарні провайдери та сервіси, що дозволяє розгортати інфраструктуру в межах різних платформ за допомогою єдиного інструменту. Крім того, завдяки його модульній архітектурі, користувачі мають можливість не тільки використовувати готові модулі, але й створювати власні, що забезпечує додаткову гнучкість та адаптивність до специфічних потреб проекту;
- ідемпотентність: Terraform гарантує, що розгортання інфраструктури буде консистентним та передбачуваним, незалежно від кількості виконань коду. Ця особливість означає, що множинні запуски Terraform на тому ж наборі конфігурацій ведуть до однакового результату, що забезпечує стабільність та знижує ризики пов'язані з випадковими змінами. Ідемпотентність є особливо важливою в контексті автоматизації та CI/CD процесів, де регулярне оновлення інфраструктури має відбуватися без непередбачуваних збоїв чи змін. Ця характеристика Terraform робить його надзвичайно надійним інструментом для управління інфраструктурою в масштабах підприємства, де консистентність та контрольованість змін є ключовими;
- управління залежностями: Terraform автоматично управляє

залежностями між ресурсами, що значно спрощує процес розгортання та управління складними інфраструктурами. Ця функціональність дозволяє Terraform визначати оптимальну послідовність створення, оновлення та видалення ресурсів, забезпечуючи цілісність і стабільність інфраструктури. Крім того, Terraform дозволяє вручну налаштовувати залежності за допомогою спеціальних вказівок у конфігураційних файлах, що надає додатковий контроль над процесом розгортання. Ця особливість є особливо цінною при управлінні багатошаровими або зв'язаними інфраструктурними компонентами, де послідовність та залежності мають критичне значення;

- модульність: Terraform дозволяє створювати та використовувати модулі, що полегшує повторне використання коду та організацію проектів. Модулі в Terraform можуть містити підготовлені набори ресурсів і конфігурацій, які можна легко інтегрувати в різні проекти, забезпечуючи стандартизацію та ефективність. Це дозволяє розробникам відокремлювати різні аспекти інфраструктури, такі як мережева інфраструктура, сервери або політики безпеки, в окремі модулі, які потім можуть бути використані в різних середовищах або проектах. Такий підхід сприяє ефективному управлінню кодом, знижує ризик помилок через повторне використання перевірених компонентів та спрощує процес оновлення інфраструктури. Крім того, Terraform Registry пропонує широкий вибір готових модулів, які розроблені спільнотою та можуть бути використані для швидкого створення складних інфраструктур.

Процес розгортання з Terraform включає:

- розробка шаблону: написання коду Terraform включає детальне визначення потрібних ресурсів та їх конфігураційних параметрів. Код Terraform, написаний у мові HCL (HashiCorp Configuration Language), дозволяє чітко та точно описати інфраструктуру, від простих об'єктів, таких як віртуальні машини, до складних конфігурацій, що включають мережі, політики безпеки, бази даних тощо. Розробка шаблону також може включати визначення вихідних даних і змінних, що забезпечує гнучкість та можливість

налаштування;

- ініціалізація: команда `terraform init` готує робоче середовище до використання. Це включає завантаження та інсталяцію вказаних провайдерів (наприклад, провайдера Azure), які Terraform використовуватиме для взаємодії з хмарними сервісами. Ініціалізація також завантажує та налаштовує модулі, зазначені в конфігурації, та підготовлює локальний кеш стану Terraform;

- планування та валідація: `terraform plan` створює та відображає виконуваний план, який деталізує зміни, що будуть застосовані до інфраструктури. Це дозволяє перевірити, що код Terraform не містить помилок та що плановані дії відповідають очікуванням, перш ніж робити будь-які зміни в реальному середовищі. Цей етап також допомагає уникнути непередбачених наслідків та забезпечує додатковий рівень безпеки;

- розгортання: коли план розгортання перевірено та затверджено, команда `terraform apply` реалізує описану конфігурацію в хмарному середовищі. Цей процес включає створення, модифікацію або видалення ресурсів відповідно до визначених у конфігурації специфікацій. Terraform веде облік поточного стану інфраструктури, забезпечуючи синхронізацію опису інфраструктури з її реальним станом;

- оновлення та управління: Terraform полегшує процес оновлення інфраструктури. При зміні конфігураційного файлу та виконанні `terraform apply`, Terraform автоматично виявляє, які компоненти інфраструктури потребують оновлення, мінімізуючи вплив на існуючі ресурси. Це дозволяє ітеративно управляти інфраструктурою, поступово вносячи необхідні зміни та оновлення.

Розглянемо застосування Terraform для створення такої ж інфраструктури, як і у випадку з Bicep та ARM Templates — веб-додатку, бази даних SQL Server, Storage Account та Key Vault в Azure. За допомогою Terraform можна ефективно управляти всіма аспектами створення та конфігурації цих ресурсів. Крім того, Terraform дозволяє легко інтегрувати

вихідні дані, такі як connection string для Storage Account, та зберігти їх в Key Vault.

Застосування Terraform в цьому контексті не лише забезпечує ефективне та автоматизоване управління хмарною інфраструктурою, але й сприяє підвищенню гнучкості, безпеки та масштабованості рішень.

Лістинг 4.3 – Код розгортання наведеної інфраструктури за допомогою Terraform

```
provider "azurerm" {
  features {}
}

variable "location" {
  default = "East US"
}

variable "webAppName" {}
variable "sqlServerName" {}
variable "sqlDatabaseName" {}
variable "storageAccountName" {}
variable "keyVaultName" {}

resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = var.location
}

resource "azurerm_app_service_plan" "example" {
  name                = "${var.webAppName}-service-plan"
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name

  sku {
    tier = "Standard"
    size = "S1"
  }
}

resource "azurerm_app_service" "example" {
  name                = var.webAppName
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  app_service_plan_id = azurerm_app_service_plan.example.id
}

resource "azurerm_sql_server" "example" {
```

```
  name                          = var.sqlServerName
  resource_group_name           =
azurerm_resource_group.example.name
  location                      =
azurerm_resource_group.example.location
  version                       = "12.0"
  administrator_login           = "sqladmin"
  administrator_login_password = "Pa$$w0rd123"
}

resource "azurerm_sql_database" "example" {
  name                = var.sqlDatabaseName
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location
  server_name         = azurerm_sql_server.example.name
  collation           = "SQL_Latin1_General_CP1_CI_AS"
  edition             = "Basic"
}

resource "azurerm_storage_account" "example" {
  name                      = var.storageAccountName
  resource_group_name       = azurerm_resource_group.example.name
  location                  =
azurerm_resource_group.example.location
  account_tier              = "Standard"
  account_replication_type = "LRS"
}

resource "azurerm_key_vault" "example" {
  name                = var.keyVaultName
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
  sku_name            = "standard"
  tenant_id           =
data.azurerm_client_config.current.tenant_id
}

resource "azurerm_key_vault_secret" "example" {
  name         = "storageAccountKey"
  value        =
azurerm_storage_account.example.primary_access_key
  key_vault_id = azurerm_key_vault.example.id
}
```

4.4 Порівняння програмних реалізацій

Автоматичне розгортання інфраструктури як коду (IaC) є ключовим компонентом сучасного управління хмарними сервісами, і інструменти, такі як ARM Templates, Bicep, та Terraform, є фундаментальними у цьому процесі.

Кожен з цих інструментів має свої унікальні особливості та переваги.

ARM Templates: це нативний інструмент для Azure, який дозволяє детально описати інфраструктуру через JSON шаблони. Він миттєво підтримує нові ресурси Azure та має вбудовану інтеграцію зі специфічними для Azure механізмами. Хоча ARM Templates є дуже потужним інструментом розгортання, його синтаксис може бути складним та вимагати часу для розуміння. Відмінною особливістю є також автоматичне управління станом. Однак, ARM Templates не забезпечує прямого доступу до властивостей ресурсів, на відміну від деяких інших інструментів.

Bicep: розроблений Microsoft як спрощена альтернатива ARM Templates, Bicep використовує більш зрозумілий синтаксис. Він трансформується в ARM шаблони, тому підтримує всі можливості Azure і є хорошим вибором для розробників, які шукають баланс між функціональністю і легкістю використання. Він трансформується в ARM шаблони, забезпечуючи повну підтримку можливостей Azure. Особливо важливою характеристикою Bicep є його здатність миттєво підтримувати нові ресурси Azure, що робить його ідеальним вибором для розробників, які шукають ефективний баланс між функціональністю та легкістю використання.

Terraform: це найбільш універсальний інструмент, який підтримує не тільки Azure, але й багато інших хмарних провайдерів. Terraform має декларативний синтаксис і дозволяє управляти інфраструктурою через його стан. Це робить Terraform ідеальним для розгортання та управління інфраструктурою в мультихмарних середовищах. Однією з ключових переваг Terraform є прямий доступ до властивостей ресурсів. Проте, слід зазначити, що Terraform не має інтеграції з деякими специфічними для Azure механізмами, що може бути важливим аспектом при виборі інструменту для певних проектів.

В контексті надання прикладів для конкретної інфраструктури, ми бачимо, що кожен інструмент має свої особливості у термінах синтаксису та

підходу до управління ресурсами. ARM Templates та Bicep зосереджені на Azure і надають найбільшу сумісність та підтримку для ресурсів Azure. Terraform, з іншого боку, пропонує більшу гнучкість і можливість управління ресурсами на різних хмарних платформах. Найкоротший код опису вигаданої інфраструктури серед усіх мав Terraform. Також Terraform представив найлегший спосіб витягнути connection string зі Storage Account, завдяки тому, що він має прямий доступ до властивостей ресурсів. Але його відсутність інтеграції зі специфічними для Azure механізмами могла спричинити проблеми при розгортанні хмарної інфраструктури, захищеної Azure Policy.

Вибір між цими інструментами залежить від конкретних потреб проекту, середовища, в якому він розгортається, та вимог до управління інфраструктурою. Ідеальний інструмент для одного проекту може не бути оптимальним для іншого, тому важливо оцінити вимоги та можливості перед вибором інструмента для автоматичного розгортання інфраструктури.

ВИСНОВКИ

У роботі було досліджені методи та засоби автоматичного розгортання інфраструктури як коду для хмарних сервісів.

В якості методів та засобів автоматичного розгортання інфраструктури як коду для хмарних сервісів були досліджені популярні інструменти автоматичного розгортання інфраструктури як коду провідних провайдерів хмарних платформ, а саме: AWS CloudFormation, Azure Resource Manager Templates, Google Cloud Deployment Manager, IBM Cloud Schematics та Oracle Cloud Infrastructure Resource Manager.

Практична реалізація завдання автоматичного розгортання інфраструктури як коду була досягнута за допомогою найбільш функціональних засобів, таких як Azure Resource Manager Templates, доменно-орієнтованої мови Bicep та мультиплатформного Terraform. За результатами досліджень можна зробити наступний висновок:

Кожен з цих інструментів має свої унікальні особливості та переваги для автоматизації розгортання інфраструктури. ARM виявився гіршим за Bicep. Bicep відрізняється тим, що пропонує більш чистий та зрозумілий синтаксис для Azure, що полегшує читання та написання коду. Тоді як Terraform виступає як універсальне рішення, яке підходить для різних хмарних платформ, забезпечуючи гнучкість у мультихмарних середовищах. Вибір між Bicep та Terraform для роботи в Azure може базуватися на критеріях, таких як вподобання у зрозумілості коду та необхідність універсальності та гнучкості для інтеграції з іншими хмарними сервісами.

Результати досліджень опубліковані у науковій статті «Особливості автоматичного розгортання інфраструктури як коду для хмарних сервісів», а також представлені на міжнародних науково-технічних конференціях: «Комп’ютерні інтелектуальні системи та мережі» та «Молоді вчені-2023. Від теорії до практики».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Guerriero M., Michele G. Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In: 2019 IEEE international conference on software maintenance and evolution (ICSME). IEEE, 2019. p. 580-589.

2. Rahman O., Akond J., Rezvan A. A systematic mapping study of infrastructure as code research. Information and Software Technology, 2019, 108: p. 65-77.

3. Rahman O., Akond J. Gang of eight: A defect taxonomy for infrastructure as code scripts. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020. p. 752-764.

4. Riti K. Pierluigi A. Infrastructure as Code. Beginning HCL Programming: Using Hashicorp Language for Automation and Configuration, 2021, p. 65-78.

5. Cloud Spending Growth Rate Slows But Q4 Still Up By $10 Billion from 2021; Microsoft Gains Market Share [Електронний ресурс]. – Режим доступу: https://www.srgresearch.com/articles/cloud-spending-growth-rate-slows-but-q4-still-up-by-10-billion-from-2021-microsoft-gains-market-share

6. Red Hat. What are cloud services? [Електрон. ресурс]. – Режим доступу: https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services

7. What is Infrastructure as Code? [Електронний ресурс] // Mike Jacobs, Ed Kaim. – 2021. – Режим доступу: https://docs.microsoft.com/enus/devops/deliver/what-is-infrastructure-as-code

8. What Is Infrastructure as Code? How It Works, Best Practices, Tutorials. [Електронний ресурс]. – Режим доступу: https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials

9. Infrastructure as Code. Microsoft Ignite. [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/azure/cloud-adoption-

framework/ready/considerations/infrastructure-as-code

10. Копцев О. О. Особливості автоматичного розгортання інфраструктури як коду для хмарних сервісів / О. О. Копцев, В. О. Мартовицкий // Системи управління, навігації та зв'язку. Збірник наукових праць. – Полтава: ПНТУ, 2024. – preprinting

11. Копцев О. О. Важливість тестування програмного продукту при використанні методології DevOps: матеріали XVI Всеукр. наук.-практ. WEB конф. аспірантів, студентів та молодих вчених «Комп'ютерні інтелектуальні системи та мережі» KICM-2023, м. Кривий Ріг, 21-23 березня 2023 р., Кривий Ріг, 2023. С. 233.

12. Копцев О. О. Особливості тестування програмного продукту при використанні практик DevOps: матеріали Всеукр. конф. «Молоді вчені-2023. Від теорії до практики», м. Дніпро, 23 березня 2023 р., Дніпро, 2023. С. 154.