Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

ТЕХНОЛОГІЯ АВТОМАТИЗАЦІЇ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ В ХМАРНОМУ СЕРЕДОВИЩІ

Виконав студент 2 курсу, групи 2КІ-22м
спеціальності 123 — Комп'ютерна інженерія
Шевченко О.В.
Керівник к.т.н., проф. каф. ОТ
Захарченко С. М.
"07" грудня 2023р.
Опонент доктор філософії, доцент каф. МБІС
Салієва О.В.
"08" грудня 2023р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.

"11" 12 2023 р.

ВНТУ 2023

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Галузь знань — Інформаційні технології

Освітній рівень — магістр

Спеціальність — 123 Комп'ютерна інженерія

Освітньо-професійна програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри обчислювальної техніки

О.Д. Азаров

"26" вересня 2023 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту **Шевченку Олександру Вікторовичу**

1 Тема роботи «Технологія автоматизації розгортання інфраструктури в хмарному середовищі» керівник роботи Захарченко Сергій Михайлович професор, затверджено наказом вищого навчального закладу від 18.09.2023 року № 247

2 Строк подання студентом роботи 12.12.2023 р.

3 Вихідні дані до роботи: ресурси розташовані у хмарному сервісі, віртуальний сервер доступу до інфраструктури, віртуальні сервери в кількості 10 шт, 3 віртуальні сервери для формування Managed instance group, одна віртуальна мережа, дві віртуальні підмережі, один балансувальник мережевого навантаження, один NAT.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз технологій автоматизованого розгортання інфраструктури в хмарних середовищах, дослідження методів розгортання

інфраструктури, розробка технології автоматизованого розгортання інфраструктури в хмарному середовищі, економічна частина.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): модель графа залежностей ресурсів, схема залежностей інфраструктурних компонентів, життєвий цикл інфраструктури, алгоритм управління інфраструктурою гілка Create, алгоритм управління інфраструктурою гілка Read, алгоритм управління інфраструктурою гілка Update, алгоритм управління інфраструктурою гілка Delete.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Захарченко Сергій Михайлович к.т.н., проф. каф. ОТ	1.09.2023	1.09.2023
4	Небава Микола Іванович проф., к.е.н	1.09.2023	1.09.2023

7 Дата видачі завдання 19.09.2023р.

8 Календарний план виконання МКР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів МКР	Строк виконання	Підпис
1	Постановка задачі	10.09.2023	
2	Огляд наявних рішень	25.09.2023	
3	Дослідження архітектури Google Cloud Platform та HashiCorp Terraform модулів	05.10.2023	
4	Розробка Infrastructure as Code та розгортання в Google Cloud Platform	12.10.2023	
5	Розрахунок економічної частини	19.10.2023	

Продовження таблиці 2

6	Оформлення пояснювальної записки	26.10.2023	
7	Виконання магістерської кваліфікаційної роботи	02.11.2023	
8	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	09.11.2023	
9	Підписи супроводжувальних документів у керівника, опонента, нормо-контролера	16.11.2023	
10	Перевірка «Антиплагіат»	23.11.2023	
11	Попередній захист	30.11.2023	

Студент Шевченко Олександр Вікторович

Керівник к.т.н., проф. Захарченко Сергій Михайлович

АНОТАЦІЯ

УДК 004.75

Шевченко О.В. Технологія автоматизації розгортання інфраструктури в хмарному середовищі. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна Інженерія, Вінниця: ВНТУ, 2023 — 121с. На укр. мові. Бібліогр.: 36 назв; рис.: 39; табл. 17.

У роботі розглянуто технології автоматизації розгортання інфраструктури в хмарному середовищі. Проведений аналіз сучасних підходів розгортання інфраструктури, розглянуто інструменти IaC, вибрано оптимальну технологію програмного створення інфраструктури в хмарному середовищі. Розроблено програмне рішення, з використанням хмари та мови конфігурації Hashicorp Configuration Language (HCL).

Ключові слова: IaC, Terraform, HCL, автоматизація, інфраструктура, хмарне середовище, ефективність, безпека, управління інфраструктурою.

ABSTRACT

УДК 004.75

Shevchenko Olekcandr. "Technology of Automated Deployment of Infrastructure in Cloud Environment". Master's qualification work in specialty 123 — Computer Engineering, Vinnytsia: VNTU, 2023 - 121 pages. In Ukrainian language. Bibliography: 36 titles; figures: 39; tables: 17.

The work discusses technologies for automating the deployment of infrastructure in a cloud environment. An analysis of modern approaches to infrastructure deployment is conducted, tools for IaC are considered, and the optimal technology for software creation of infrastructure in the cloud environment is selected. A software solution is developed, using the cloud and Hashicorp Configuration Language (HCL) for configuration.

Keywords: IaC, Terraform, HCL, automation, infrastructure, cloud environment, efficiency, security, infrastructure management.

ЗМІСТ

ВСТУП

Інфраструктура як код (IaC) дозволяє надавати та підтримувати обчислювальну інфраструктуру за допомогою коду замість ручних процесів та налаштувань. Будь-яке середовище серверних додатків потребує безліч компонентів інфраструктури, таких як операційні системи, підключення до баз даних та сховища і т.д..

Ручне управління інфраструктурою займає багато часу, і при ньому є ризик виникнення помилок. Інфраструктура як код (IaC) дозволяє визначити бажаний стан інфраструктури. Завдяки автоматизованому управлінню інфраструктурою, розробники можуть зосередитися на розробці та вдосконаленні додатків, а не на керуванні середовищем. Організації використовують інфраструктуру як код для контролю витрат, зниження ризиків та швидкого реагування на нові бізнес-можливості.

Сучасні організації, незалежно від їх масштабів та галузей діяльності, все більше спрямовані на використання хмарних інфраструктур для забезпечення високої доступності, масштабованості та зручності управління своїми обчислювальними ресурсами. Однак, для ефективного використання хмарних платформ необхідно забезпечити автоматизацію розгортання та управління інфраструктурою.

Метою роботи є розробити програмні модулі керування інфраструктурою з використанням HashiCorp Configuration Language (HCL) в хмарному середовищі.

Для досягнення мети необхідно розв’язати такі **задачі**:

— проаналізувати наявні підходи до автоматизації розгортання інфраструктури в хмарних середовищах;

— розробити план та методику автоматизації процесу розгортання і управління інфраструктурою в хмарному середовищі;

— провести практичні дослідження та експерименти з використанням розроблених модулів для перевірки ефективності та переваг IaC;

— оцінити вплив використання автоматизації на ефективність керування хмарною системою.

Об’єктом дослідження є процес керування інфраструктурою в хмарному середовищі.

Предметом дослідження є програмні модулі керування інфраструктурою в хмарному середовищі, створення, налаштування та управління хмарними ресурсами.

Новизна вдосконалено методи розгортання інфраструктури в хмарному середовищі шляхом створення програмного коду котрий керує цим процесом, що дозволило зменшити витрати часу та мінімізувати вплив людського чинника при повторних розгортаннях.

Практична значення полягає в розробці методики формування інструкцій для розгортання інфраструктури в хмарному середовищі та практичних рекомендацій щодо розгортання.

Апробацію результатів наукової роботи було проведено на науковій конференції : «Молодь в науці: дослідження, проблеми, перспективи (МН — 2024)», доповідь на тему «Технологія автоматизації розгортання інфраструктури в хмарному середовищі».

1 АНАЛІТИЧНИЙ ОГЛЯД ТЕХНОЛОГІЙ АВТОМАТИЗАЦІЇ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ

1.1 Інформаційна інфраструктура

Інформаційна інфраструктура — сукупність різноманітних інформаційних (автоматизованих) систем, інформаційних ресурсів, телекомунікаційних мереж і каналів передачі даних, засобів комунікацій і управління інформаційними потоками, а також організаційно-технічних структур, механізмів, що забезпечують їх функціонування.

Традиційна інфраструктура, апаратне та програмне забезпечення. Наприклад, це можуть бути різноманітні сервери та стаціонарні комп'ютери. Все обладнання розташоване під одним дахом, що полегшує доступ до необхідної інформації та обчислювальних потужностей.

Ця інфраструктура вимагає багато вільного місця на місці розгортання, а також потужності самого обладнання. Як наслідок, розробка та встановлення такого типу інфраструктури обходиться дорожче [1,2].

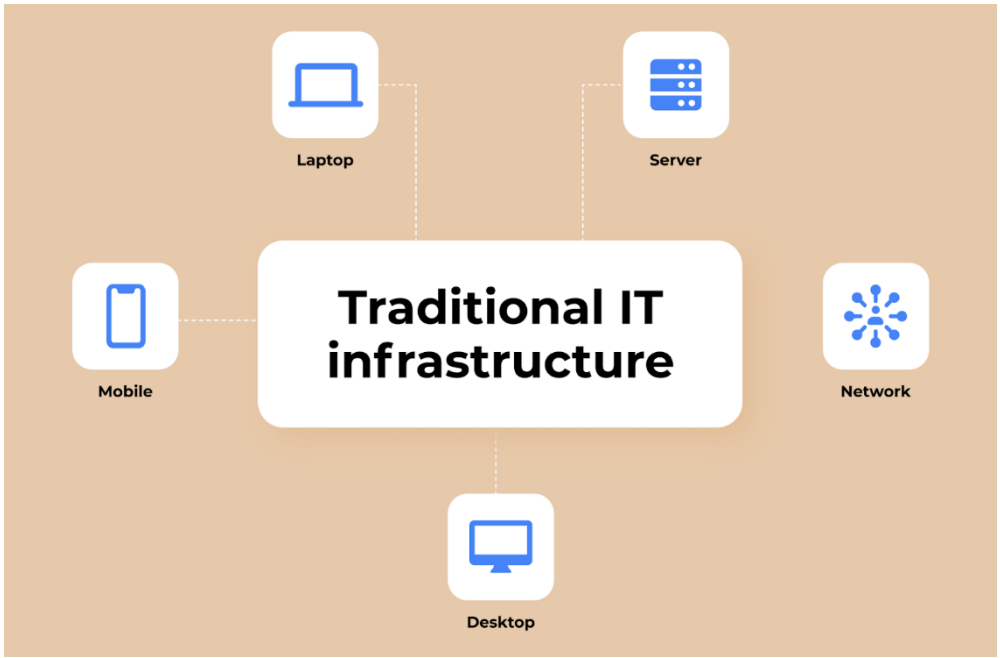

Рисунок 1.1 — Традиційна інфраструктура

Переваги традиційної інфраструктури:

— разом із системою ви отримуєте спеціальну команду, яка повністю задіяна в управлінні інфраструктурою;

— разом з оновленням інфраструктури вам також необхідно оновлювати програмне забезпечення, що дозволяє завжди бути на вістрі нових технологічних рішень;

— ви маєте можливість працювати з різними типами програмного забезпечення.

Хмарна інфраструктура — це сукупність апаратних (hardware) та програмних (software) компонентів, що необхідні для реалізації хмарних обчислень. Зазвичай цей термін використовується в контексті послуг хмарних провайдерів, у яких можна орендувати хмару під конкретні завдання бізнесу [1].

За оцінками IDC, International Data Corporation (IDC) — міжнародна компанія, постачальник маркетингових досліджень, консультаційних послуг, організатор конференцій в області інформаційних технологій, телекомунікацій і споживчої електроніки, до 2025 року 85% організацій підвищать стійкість на 35% лише за рахунок використання програмного забезпечення та хмарних інфраструктур.

Як наслідок, вам не доведеться витрачати купу грошей на обладнання та центри обробки даних, які можуть обслуговувати внутрішню IT-інфраструктуру вашого бізнесу. Замість цього ви можете знайти постачальника хмарних послуг і орендувати хмарне сховище, необхідне для перенесення туди всіх ваших внутрішніх процесів (включаючи цінні дані, додатки та багато іншого).

Крім того, ви можете розмістити свою IT-інфраструктуру на одному хмарному або декількох хмарних провайдерів. В результаті ваш бізнес отримає всі ці переваги від розробки хмарних додатків:

— усі дані зберігатимуться у хмарі, але якщо буде потреба, користувачі зможуть зберігати їх на своїх пристроях, щоб мати доступ до них в офлайн-режимі;

— співробітники можуть користуватися таким додатком на будь-якому пристрої з доступом будь-де;

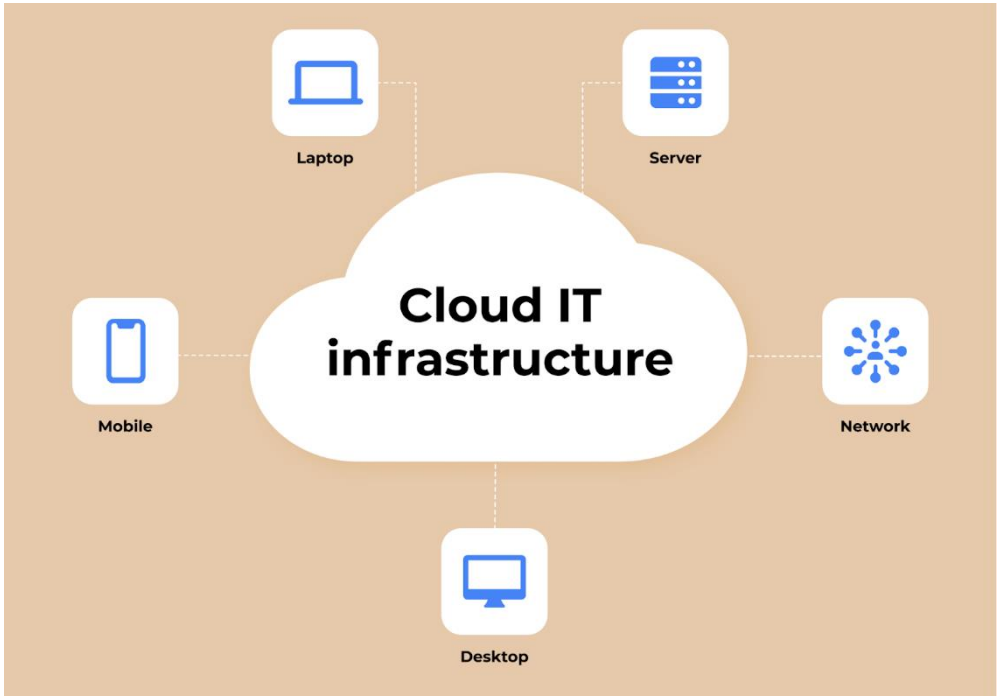

Рисунок 1.2 — Хмарна інфраструктура

Переваги хмарної IT-інфраструктури, велика гнучкість і масштабованість, гнучкість і масштабованість важливі, якщо ваша компанія прагне до швидкого зростання. А хмарні структури не мають обмежень на простір для зберігання даних і набагато більше потужності для обчислень. Крім того, набагато простіше масштабувати всю структуру вгору і вниз, а також збільшувати або зменшувати кількість серверів, якщо вам це потрібно [1].

Можливості автоматизації, впроваджуючи хмарні технології у свій бізнес, ви позбавляєтесь головного болю, пов'язаного з управлінням обладнанням та питаннями безпеки. Всі ці операції передаються на аутсорсинг постачальнику, який надає вам хмарну структуру, в той час як ви можете зосередитися на інших важливих аспектах бізнесу.

Ви можете бути здивовані, наскільки економічно вигідною є хмарна IT-інфраструктура. Це головним чином тому, що ви платите лише за конкретну послугу, якою користуєтесь. Вам не потрібно витрачати гроші на обладнання, управління, оновлення та інші аспекти, якими зараз займається постачальник [1, 2].

1.2 Інфраструктура як код (Infrastructure as Code)

Це управління та створення інфраструктури за допомогою коду, а не за допомогою ручних процесів.

За допомогою Infrastructure as Code (IaC) створюються файли конфігурації, які містять специфікації вашої інфраструктури, що полегшує редагування та розповсюдження конфігурацій. Це також гарантує, що ви щоразу створюєте одне й те саме середовище. Кодуючи та документуючи ваші специфікації конфігурації, IaC допомагає керувати конфігурацією та допомагає вам уникнути незадокументованих, спеціальних змін конфігурації [3].

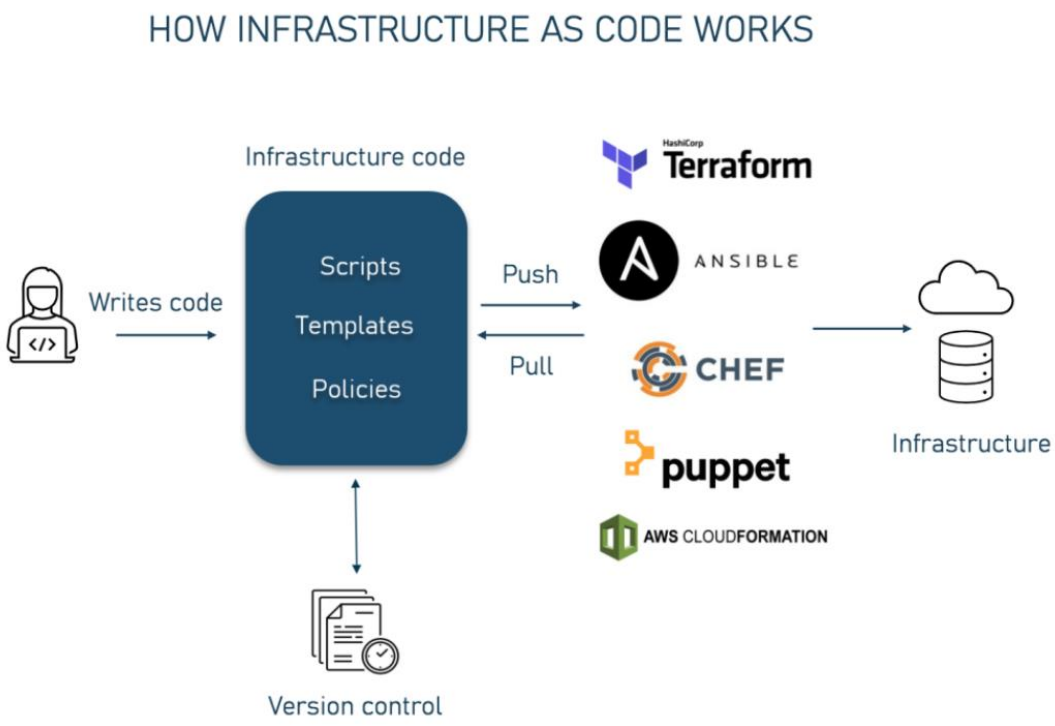

Рисунок 1.3 — Принцип роботи інфраструктура як код

Контроль версій є важливою частиною IaC, і ваші файли конфігурації повинні перебувати під їх керуванням, як і будь-який інший файл вихідного коду програмного забезпечення, цим повинна займатись система контролю версій. Це система, що записує зміни у файл або набір файлів протягом деякого часу, так що ви зможете повернутися до певної версії пізнише.

Розгортання вашої інфраструктури як коду також означає, що ви можете розділити свою інфраструктуру на модульні компоненти, які потім можна комбінувати різними способами за допомогою автоматизації.

Автоматизація надання інфраструктури за допомогою IaC означає, що розробникам не потрібно вручну надавати та керувати серверами, операційними системами, сховищем та іншими компонентами інфраструктури кожного разу, коли вони розробляють або розгортають програму.

Існує 2 способи підходу до IaC:

— декларативний;

— імперативний.

Декларативний підхід визначає бажаний стан системи, включно з тим, які ресурси вам потрібні та які властивості вони повинні мати, а інструмент IaC налаштує це для вас.

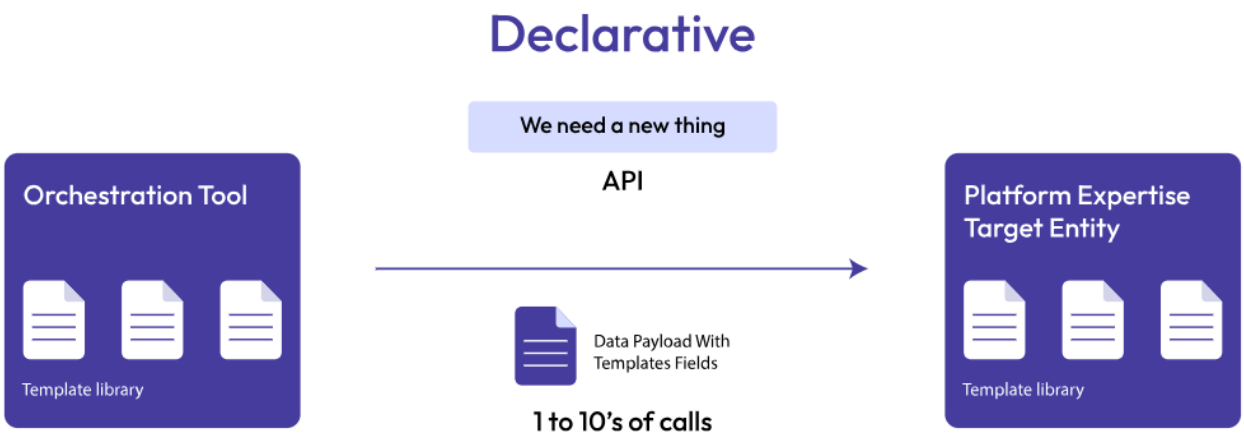

Рисунок 1.4 — Декларативний підхід

Імперативний підхід визначає конкретні команди, необхідні для досягнення бажаної конфігурації, і ці команди потім потрібно виконувати в правильному порядку (рисунок 1.5).

Багато інструментів IaC використовують декларативний підхід і автоматично надають бажану інфраструктуру. Якщо ви внесете зміни до бажаного стану, декларативний інструмент IaC застосує ці зміни за вас. Необхідний інструмент вимагатиме від вас з'ясувати, як ці зміни слід застосувати. Інструменти IaC часто

можуть працювати в обох підходах, але, як правило, віддають перевагу одному підходу над іншим [3,4].

Рисунок 1.5 — Імперативний підхід

Таблиця 1.1 — Основні переваги впровадження IaC

Процес	Опис
1	2
Автоматизація процесів	Управління складними хмарними середовищами вимагає технічних навичок та фінансових ресурсів, що може навантажити бюджет вашої компанії. В таких випадках використання IaC виявляється розумним рішенням, адже воно дозволяє спростити управління хмарною інфраструктурою, залучаючи лише декілька інженерів.

Продовження таблиці 1.1

1	2
Повторюваність розгортань	Сучасні тенденції показують, що більшість підприємств перейшли від кількох розгортань щомісяця до сотень розгортань щодня. З такою швидкістю безсумнівно, існує потреба в надійній та автоматизованій системі управління інфраструктурою.
Вимоги до масштабування	Коли ви визначаєте вимоги до інфраструктури як код, масштабувати вгору та вниз буде простіше з мінімальними інвестиціями часу та коштів.
Декларативна парадигма	IaC робить процес створення інфраструктури значно легшим. Після адаптації під ваші робочі навантаження, не потрібно переглядати об'ємну документацію або постійно вносити зміни в стан інфраструктури. Використовуючи декларативний підхід Infrastructure as Code, вам лише необхідно визначити бажаний стан системи та підтримувати конфігурацію в цьому стані, зосереджуючись на управлінні контролерами
Покращена співпраця	Так як IaC функціонує на принципах кодування, він забезпечує додаткові можливості для командної роботи завдяки використанню систем контролю версій.
Відповідність найкращим практикам	IaC надає можливість досягнути високого рівня доступності та відповідати стандартам безпеки, при цьому знижуючи ризики, що виникають під час експлуатації системи.

1.3 Програмні засоби для створення інфраструктури

Інструменти розгортання інфраструктури можна розділити на два основних типи наведені в таблиці 1.2 та в таблиці 1.3.

Native tools, які пропонуються хмарними провайдерами і зазвичай інтегровані безпосередньо в їх платформи [5].

Таблиця 1.2 — Інструменти хмарних провайдерів (Native tools)

Інструмент	Короткий опис
1	2
Google Cloud Deployment Manager	Інструмент для управління інфраструктурою Google Cloud, який використовує мову програмування високого рівня для автоматизації створення та управління хмарними ресурсами за допомогою сценаріїв YAML і Python. Цей інструмент зручний для розробників, оскільки дозволяє повторно використовувати код для послідовного розгортання і оцінки впливу змін до їх реалізації. Він також інтегрований з екосистемою Google, що усуває потребу в додатковому програмному забезпеченні для налаштування.
AWS Cloud Formation	AWS CloudFormation ефективно управляє ресурсами AWS, спрощуючи управління інфраструктурою та зменшуючи час на запуск застосунків. Він дозволяє швидко реплікувати інфраструктуру в різних регіонах, забезпечуючи високу доступність. Також CloudFormation забезпечує легке відстеження та управління змінами в інфраструктурі, підвищуючи безпеку та ефективність оновлень.

Продовження таблиці 1.1

1	2
Azure Resource Manager	Інструментом від Microsoft є важливим IaC рішенням для керування інфраструктурою Azure, використовуючи шаблони для управління залежностями та інфраструктурою. Він полегшує контроль доступу до служб і ресурсів за допомогою ролевого контролю доступу (RBAC) та дозволяє користувачам налаштовувати обсяг доступу через групи та підписки. ARM також надає ефективні інструменти для організації, які сприяють кращому управлінню ресурсами та групами.

Third-Party Tools це інструменти розроблені сторонніми виробниками і часто надають більшу гнучкість та підтримку різноманітних хмарних сервісів.

Таблиця 1.3 — Інструменти сторонніх виробників (Third-Party Tools)

Інструмент	Короткий опис
1	2
Terraform	Один із найбільш використовуваних інструментів інфраструктури як код (IaC) на ринку, який підтримує більшість хмарних платформ. Terraform підвищує надійність, зберігаючи узгодженість між усіма постачальниками хмарних платформ. Terraform також є інструментом з відкритим вихідним кодом із багатьма інструментами та сценаріями, що робить основу Terraform міцною.

Продовження таблиці 1.3

1	2
Ansible	Розроблений Red Hat Inc., є відкритим програмним забезпеченням для керування конфігурацією, подібним до Terraform, з комерційним розширенням, таким як Ansible Tower. Його унікальна архітектура не вимагає сервера або агента, використовуючи SSH/PowerShell для керування UNIX, Linux, Windows хостами та виконання конфігураційних завдань. Ansible є потужним інструментом автоматизації, дозволяючи відстежувати зміни та керувати серверами і службами через ansible-playbook та системи контролю версій.
Puppet	Інструмент із відкритим кодом; на відміну від інших інструментів, він дещо відрізняється. Puppet можна інтегрувати з будь-якою існуючою платформою. Це основа для багатьох конвеєрів Continuous Integration/Continuous Delivery, створених інженерами DevOps. Puppet — це мова програмування високого рівня, яка може вирішувати проблеми без процедури програміста. У Puppet також є велика спільнота для його вдосконалення та розширення.
Pulumi	Інструмент IaC, який відрізняється від інших інструментів у нашому списку тому що має підтримку таких мов програмування, як Python, JavaScript, C#, Go та Typescript. Завдяки цьому Pulumi може легко відповідати багатьом сценаріям використання DevOps і охопити більшість розробників. Pulumi також має більше доступних фреймворків та інструментів для створення та тестування інфраструктури. Однією з унікальних особливостей Pulumi є те, що пропонує підтримку хмарних гігантів, таких як AWS, CGP і хмара Azure.

Закінчення таблиці 1.3

1	2
Chef	Є одним із популярних інструментів IaC у галузі. Однак він має величезну перевагу перед іншими. Він може працювати з усіма платформами, такими як AWS, Google Cloud і Azure, дозволяючи користувачам легко перемикатися між різними хмарними платформами, не втрачаючи своїх конфігурацій IaC. Його можна масштабувати та застосувати в будь-якому існуючому конвеєрі Continuous Integration/Continuous Delivery.
Vagrant	Продукт компанії HashiCrop, того самого творця Terraform, для великомасштабного обчислювального середовища. Тим не менш, продукт в основному зосереджується на невеликих обчислювальних середовищах. Він також добре працюватиме з усіма операційними системами, такими як Windows, Linux і Mac. Vagrant використовує декларацію, яка також може добре працювати з іншими інструментами IaC, особливо якщо цей інструмент є процедурним і може забезпечити одноманітність у всій системі.

Native tools, які пропонуються хмарними провайдерами, зазвичай є інтегрованими в їхні платформи і забезпечують тісну сумісність з конкретними хмарними сервісами. Натомість, Third-Party Tools, розроблені сторонніми виробниками, часто пропонують більшу гнучкість і підтримку різноманітних хмарних сервісів. У таблицях 1.2 та 1.3 представлено порівняльний аналіз популярних інструментів IaC (Infrastructure as Code) з їх коротким описом. З урахуванням цього, можна зробити висновок, що на ринку існує достатньо інструментів для створення та керування IaC, що дозволяє користувачам вибирати рішення, що найкраще відповідає їх потребам [5,6].

Дані про різні інструменти IaC, включаючи інформацію про те, чи є вони з відкритим чи закритим кодом, які хмарні провайдери підтримуються ними,

загальну кількість учасників проекту та їх рейтенгу на GitHub, кількість доступних відкритих вихідних бібліотек для кожного інструменту, а також кількість запитань, пов'язаних з ними на популярні системі питань і відповідей для професійних програмістів та ентузіастів Stack Overflow. Результати наведено в таблиці 1.3. Варто зазначити, що інформаціє про деяких постачальників з закритим кодом, як-от CloudFormation чи Cloud Deployment Manager, недоступна.

Таблиця 1.4 — Порівняння спільноти та вкладень в інструменти IaC

Tool (інструмент)	Source (джерело)	Cloud	contributors (вкладників)	Stars (GitHub)	Libraries (бібліотеки)	Stack Overflow (віккритих тем)
Chef	open	all	640	6910	3695	8295
Puppet	open	all	571	6581	6871	3996
Polumi	open	all	1402	12723	15	327
Ansible	open	all	5328	23479	31329	12052
Terraform	open	all	1621	33019	9641	13370
Cloud Deployment Manager	closed	Google	-	-	402	9134
Cloud Formation	closed	AWS	-	-	369	7252
Resource Manager (ARM)	closed	Azure	-	-	398	5687

Деякі інструменти мають більше ніж одне сховище, наприклад, у 2017 році Terraform розділив код провайдера, специфічний для:

— Amazon Web Services;

— Google Cloud Platform;

— Microsoft Azure.

За допомогою даних, представлених у таблиці 1.4, можна визначити лідерів у сфері IaC, серед яких виділяються Terraform та Ansible. Ці інструменти переважають інших завдяки значній кількості доступних відкритих бібліотек, великій кількості учасників, які беруть участь у їх розробці, а також високій активності обговорень на форумі Stack Overflow.

Використовуючи дані з таблиці 1.4, які відображають зміни коду на GitHub, здійснені спільнотою між 2016 та 2023 роками, ми створили графічне відображення цієї інформації (рисунок 1.6). Цей графік візуалізує динаміку змін коду, внесених учасниками спільноти протягом зазначеного періоду.

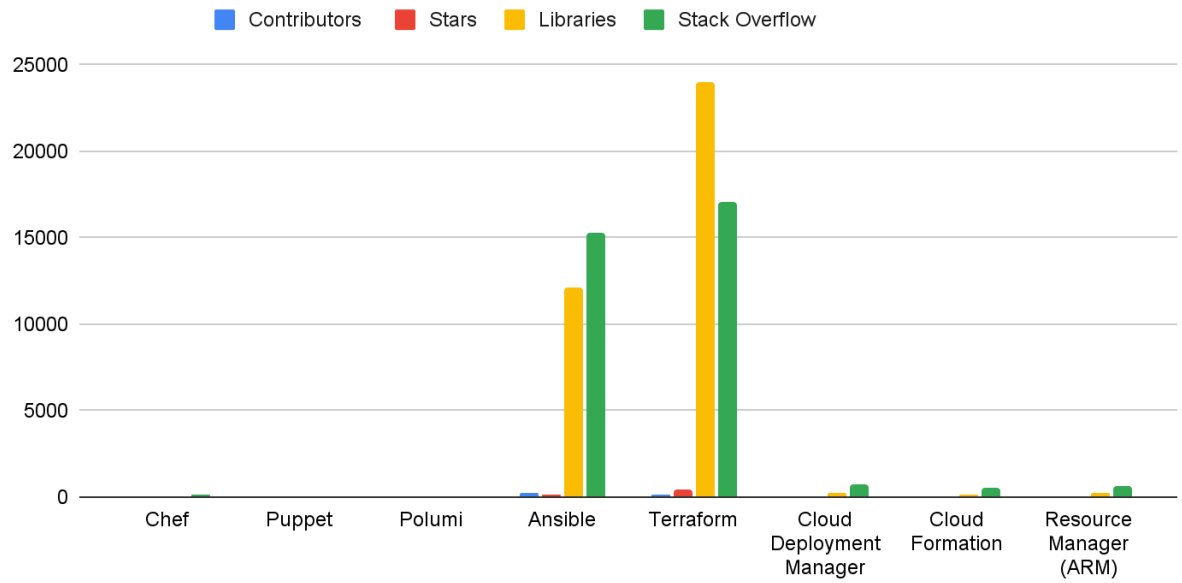

Рисунок 1.6 — Кількість змін на GitHub зроблених спільнотою з 2016 — 2023

Terraform і Ansible переживають вибухове зростання. Збільшення кількості співавторів, рейтингу, бібліотек з відкритим кодом і публікацій Stack Overflow. Обидва ці інструменти сьогодні мають великі активні спільноти, і, судячи з цих тенденцій, цілком імовірно, що в майбутньому вони стануть ще більшими.

Ще одним ключовим фактором, який слід враховувати при виборі будь-якої технології, є зрілість. А отже варто поставити наступні питання:

— чи це технологія, яка існує вже багато років, де добре зрозумілі всі моделі використання, найкращі практики, проблеми та режими збоїв?

— чи це нова технологія, де вам доведеться вивчати все з нуля?

Важливим критерієм при виборі технології є її зрілість, яка включає розуміння моделей використання, найкращих практик, виявлених проблем та сценаріїв збоїв. Таблиці 1.5 демонструє початкові дати випуску інструментів.

Таблиця 1.5 — Рік першого релізу

Tool (Інструмент)	Initial release (Початковий випуск)	Current Version (Поточна версія)
Chef	2009	23.7.1042
Puppet	2005	8.3.1
Polumi	2017	3.91.1
Ansible	2012	2.15
Terraform	2014	1.6.2
Cloud Deployment Manager (SDK)	2015	452.0.1
Cloud Formation	2011	2.13.30
Resource Manager (ARM)	2014	2.53.1

Нові технології можуть вимагати глибокого вивчення та адаптації, у той час як більш зрілі технології, такі як Terraform, вже мають усталену базу знань і досвід. З огляду на ці аспекти, Terraform виглядає більш привабливим вибором порівняно з Ansible, особливо в контексті довгострокового використання, кількості змін на GitHub та стабільності [7].

1.5 Аналіз інструментів IaC

Terraform і Ansible є двома популярними інструментами, які використовуються в області інфраструктури як код. Обидва інструменти використовуються для автоматизації підготовки та керування інфраструктурою, але вони служать різним цілям і пропонують різні можливості.

1.5.1 Інструмент IaC Ansible

Це інструмент автоматизації та керування налаштуваннями. Він призначений для автоматизації завдань, таких як встановлення та налаштування програмного забезпечення, на кількох віддалених хостах в мережі. Ansible розроблений як без-агентний, що означає, що він може керувати віддаленими системами без необхідності встановлення будь-якого програмного забезпечення на таких системах. Він використовує просту, зрозумілу людині мову під назвою YAML для опису бажаного стану систем і програм. Організації можуть ефективніше та послідовніше керувати своєю інфраструктурою за допомогою Ansible шляхом автоматизації завдань [8].

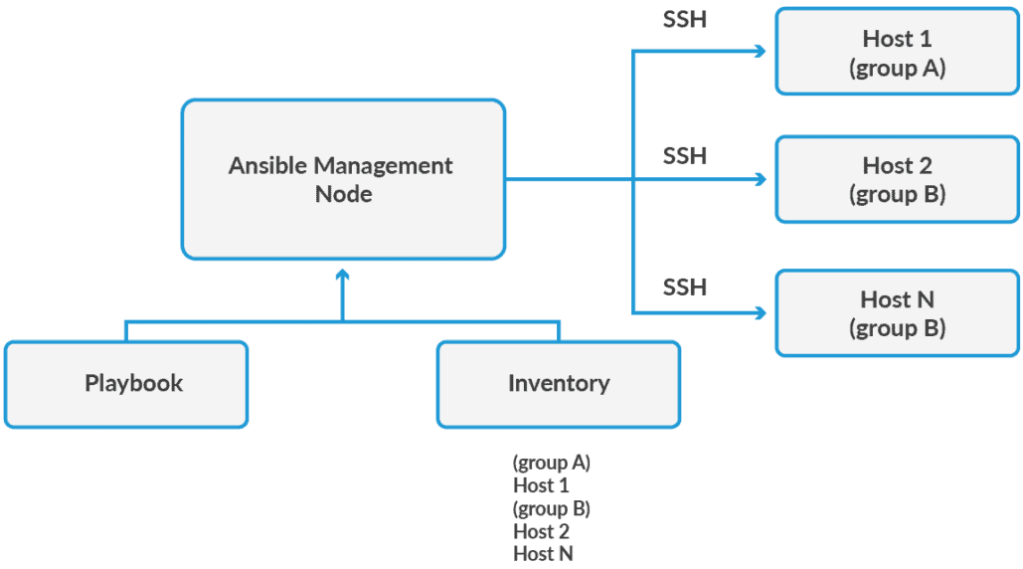

Рисунок 1.7 — Схема роботи IaC Ansible

Особливості Ansible:

— використовується для керування конфігурацією та дотримується процедурного підходу;

— працює з як з традиційними інфраструктурними платформами так із хмарними;

— дотримується ідемпотентної поведінки (метод, який можна викликати багато разів без різних результатів), що змушує його переводити вузол у той самий стан кожного разу;

— використовує інфраструктуру як системну конфігурацію коду в усій інфраструктурі;

— пропонує швидке та просте розгортання багаторівневих програм без використання агента;

— якщо код переривається, це дозволяє ввести код знову без будь-яких конфліктів з іншими викликами.

1.5.2 Інструмент IaC Terraform

Це інструмент із відкритим вихідним кодом для безпечного та ефективного створення, зміни та керування версіями інфраструктури. Будучи інструментом «Інфраструктура як код» (IaC), він дає користувачам змогу керувати та надавати інфраструктуру за допомогою процедур кодування, а не вручну. Для створення інфраструктури Terraform пропонує мову налаштування високого рівня.

Він також підтримує низку хмарних провайдерів, зокрема:

— Amazon Web Services;

— Microsoft Azure;

— Google Cloud Platform;

— та багато інших хмарних провайдерів.

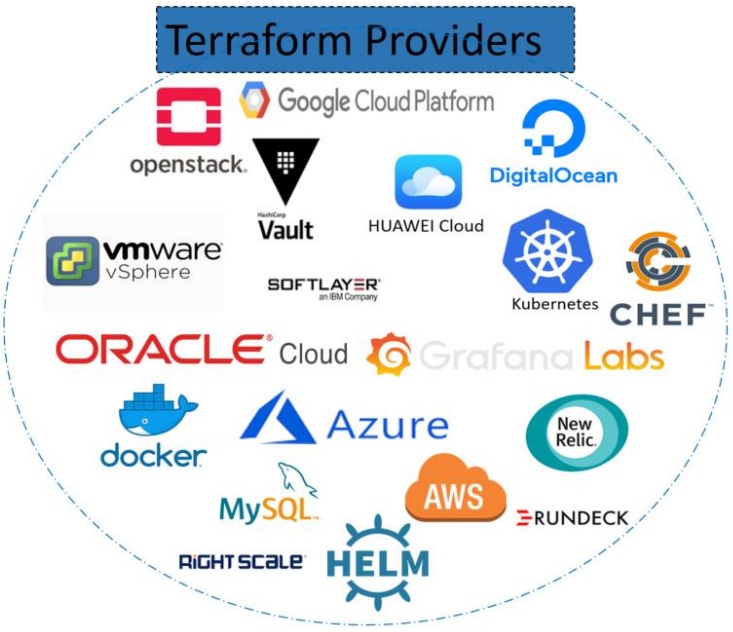

Рисунок 1.8 — Підтримка хмарних провайдерів

Організації можуть використовувати Terraform для стандартизованого, повторюваного та версійного керування своєю інфраструктурою.

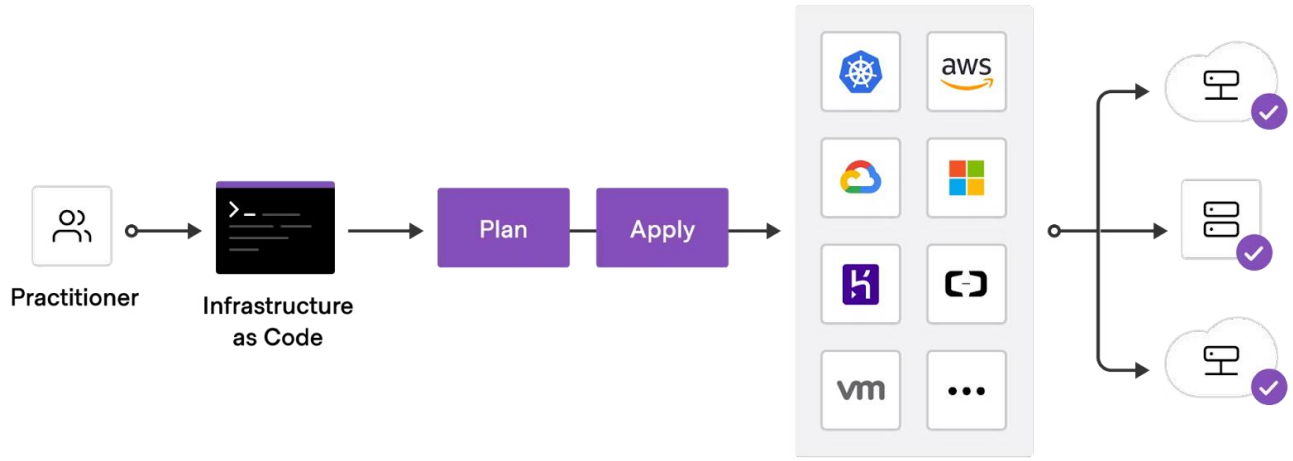

Рисунок 1.9 — Схема роботи роботи IaC Terraform

Особливості Terraform:

— дотримується декларативного підходу, що робить розгортання швидким і легким;

— зручний інструмент для відображення отриманої моделі в графічній формі;

— керує зовнішніми постачальниками послуг, такими як хмарні мережі та власні рішення;

— один із рідкісних інструментів, який пропонує створення інфраструктури з нуля, будь то публічна, приватна чи мультихмарна;

— допомагає керувати паралельними середовищами, що робить його хорошим вибором для тестування, перевірки виправлень помилок і офіційного прийняття;

— модульний код допомагає досягти узгодженості, багаторазового використання та співпраці;

— може керувати кількома хмарами для підвищення відмовостійкості.

Terraform дозволяє використовувати той самий робочий процес для керування декількома провайдерами та керування міжхмарними залежностями. Це спрощує керування для великомасштабних багатохмарних інфраструктур [8].

У великій організації ваша централізована операційна група може отримувати багато повторюваних запитів інфраструктури. Ви можете використовувати Terraform, щоб побудувати модель інфраструктури, яка дозволяє групам продуктів самостійно керувати власною інфраструктурою. Ви можете створювати та використовувати модулі Terraform, які кодифікують стандарти для розгортання та керування службами у вашій організації, дозволяючи командам ефективно розгортати служби відповідно до практик вашої організації [8].

У вас можуть бути проміжні середовища або середовища контролю якості, які ви використовуєте для тестування нових програм перед випуском їх у виробництво. Оскільки виробниче середовище стає все більшим і складнішим, стає все важче підтримувати оновлене середовище для кожного етапу процесу розробки. Terraform дозволяє швидко розгортати та виводити з експлуатації інфраструктуру для розробки, тестування, контролю якості та виробництва. Використання Terraform для створення одноразових середовищ за потреби є більш рентабельним, ніж підтримувати кожну з них нескінченно довго [9].

1.7 Мова програмування Hashicorp Configuration Language

Мова конфігурації HashiCorp (HCL) — це унікальна мова конфігурації. Вона була розроблена для використання з інструментами HashiCorp, особливо з Terraform, але HCL розширилася як більш загальна мова конфігурації. Вона візуально схожа на JSON з додатковими структурами даних та вбудованими можливостями [4].

HCL складається з трьох підмов:

— structural;

— expression;

— templates.

Structural — частина HCL відповідає за визначення структури конфігурації. Вона дозволяє користувачам описувати конфігурації у вигляді ієрархічних блоків, які можуть включати ресурси, модулі, змінні тощо. Цей аспект мови забезпечує чітку організацію та легкість управління конфігураціями.

Expression — вирази у HCL використовуються для виконання динамічних обчислень всередині конфігурацій. Це може включати арифметичні операції, доступ до даних змінних, використання функцій тощо. Вирази дозволяють створювати більш гнучкі та адаптивні конфігурації.

Templates — шаблони в HCL дозволяють вбудовувати шаблонізований текст, який може бути згенерованим або трансформованим під час виконання конфігурації. Це особливо корисно для генерації динамічних рядків або конфігураційних файлів, які мають включати змінні або вирази, що обчислюються в реальному часі [10].

HCL також можна використовувати з інструментами, крім Terraform. З часом з'явилися різні парсери, такі як Go, Java та Python, їх популярність наведенв на (рисунок 1.10).

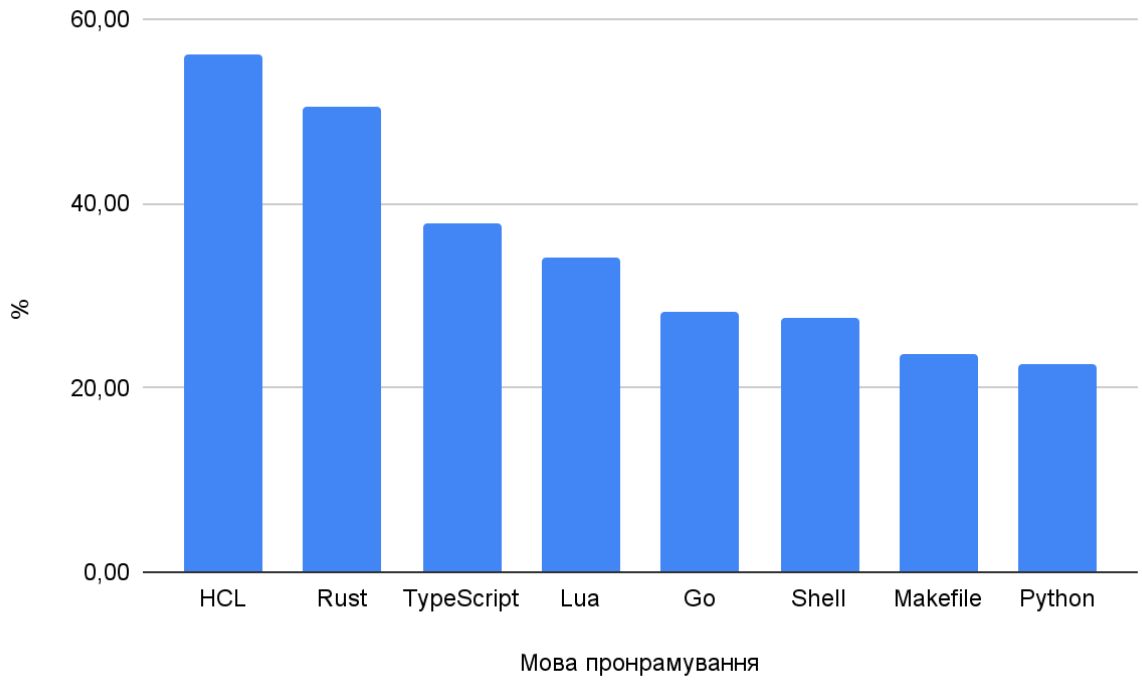

Рисунок 1.10 — Рейтинг парсерів мов програмування IaC з 2021 по 2023

Рейтинг мов програмування, які швидкими темпами набирають популярність для програмування IaC. Ця інформація представлена у щорічному звіті Octoverse.

За останній рік використання мова конфігурації Hashicorp (HCL) значно зросло. Це було зумовлено зростанням популярності інструменту Terraform і практик IaC для все більшої автоматизації розгортань [10,11].

Базовий синтаксис мови HCL наведено в таблиці1.6. з прикладами та поясненнями.

Таблиця 1.6 — Базовий синтаксис мови HCL

Команди	**Приклад**	**Пояснення**
1	2	3
Arguments Аргументи	image_id = "abc123"	Ідентифікатор перед знаком рівності є назвою аргументу, а вираз після знака рівності є значенням аргументу.

Blocks
Блоки
	resource
	"google_compute_instance"
	"example" {
	ami = "abc123"
	network_interface {
	# ...
	}
	}
		Блок має тип (**ресурс** у цьому прикладі). Кожен тип блоку визначає, скільки міток має слідувати за ключовим словом type.

Identifiers
Ідентифікатори
	resource
	"google_storage_bucket"
	"**terraform_state**" {
	name = "${var.project}-terraform-state"
	project = var.project
	location = var.region
	force_destroy = false
	}
		Назви аргументів, назви типів блоків і назви більшості специфічних для Terraform конструкцій, таких як ресурси, вхідні змінні тощо, є ідентифікаторами.

HashiCorp Configuration Language (HCL) має кілька переваг над іншими мовами IaC:

— легкість використання: HCL — це простий у вивченні і використанні мову конфігурації;

— зручність читання: HCL має чітку і зрозумілу синтаксис, що полегшує читання та розуміння конфігурацій;

— підтримка Terraform: HCL є офіційною мовою конфігурації Terraform, що означає, що він повністю підтримується Terraform;

— для невеликих або простих інфраструктурних проектів. HCL може бути кращим вибором для невеликих або простих інфраструктурних проектів, де не потрібно створювати складні конфігурації;

— для розробників, які не мають досвіду роботи з програмуванням. HCL може бути кращим вибором для розробників, які не мають досвіду роботи з програмуванням;

— для проектів, які потребують високої продуктивності. HCL може бути кращим вибором для проектів, які потребують високої продуктивності.

Приклад створення ресурсу наведений в лістинг 4.

Лістинг 4 — Створення ресурсу

```
resource "google_compute_instance" "default" {          # Block
    name = "my-server"
    machine_type = "n2-standard-2"                      # Argument
    zone = "us-central1-a"
    tags = ["foo", "bar"]
    boot_disk {                                         # Block
        initialize_params {                             # Block
            image = "debian-11"
            labels = {                                  # Block
                my_label = "value"                      # Argument
            }
        }
    }
```

```
service_account {
    email  = google_service_account.default.email
    # Argument with value as expression
    scopes = ["cloud-platform"] } }
```

Отже, HCL — це потужний інструмент, який може допомогти інженерам інфраструктури створювати та управляти складними інфраструктурними рішеннями. Його простота використання, зручність читання та ефективність роблять його хорошим вибором для широкого спектру проектів.[12]

2 РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДІВ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ

2.1 Моделювання направленого графу ресурсів залежностей

Керування залежностями в Terraform включає автоматичне відстеження порядку створення, оновлення та видалення ресурсів на основі їх взаємозалежностей.

Граф ресурсів залежностей є ключовим інструментом при проєктуванні обчислювальної інфраструктури, оскільки він відображає взаємозв'язки між різними елементами. Використання графа залежностей дозволяє Terraform ефективно управляти складними інфраструктурами, забезпечуючи правильний порядок виконання операцій. Такі залежності мають деревоподібну структуру, що спрощує візуалізацію та розуміння взаємозв'язків між ресурсами. Кожен вузол у графі залежностей представляє окремий ресурс, а ребра вказують на залежності між цими ресурсами. Terraform використовує цей граф для визначення транзитивних відношень між ресурсами, що дозволяє автоматизувати процеси розгортання. Неявні залежності в Terraform створюються через посилання між ресурсами, які визначаються в коді конфігурації.

Залежності між ресурсами, як наприклад віртуальні машини та образи дисків, є критично важливими для забезпечення стабільності та надійності інфраструктури. Автоматичне керування ними зменшує ризик помилок, які можуть виникнути при ручному управлінні порядком створення ресурсів. Така організація ресурсів дає можливість швидше реагувати на зміни вимог та умов проекту. Додатково, це сприяє ефективній координації між різними командами, що працюють над проектом. Ясність у структурі залежностей також спрощує процес документування та передачі знань в команді.

Оптимізація процесів через граф залежностей дозволяє зменшити час на впровадження змін та оновлень. Ефективне використання ресурсів сприяє зниженню витрат, що особливо важливо для великих та складних проектів. Це також дозволяє краще враховувати потреби замовників, адаптуючи інфраструктуру

під конкретні вимоги. Завдяки гнучкості Terraform, можливо швидко реагувати на зміни у сфері технологій, підтримуючи інфраструктуру на передовому рівні [12].

Отже, змоделюємо та проаналізуємо такий граф (рисунок 2.1).

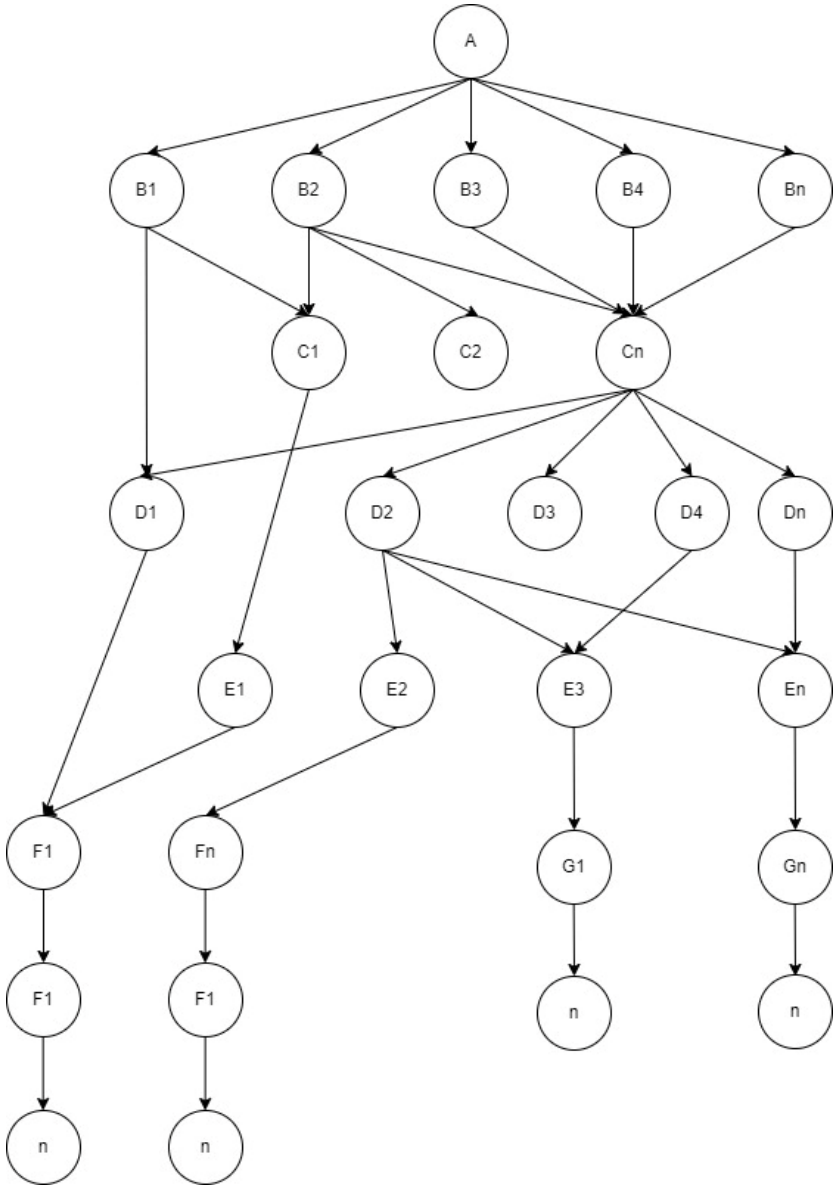

Рисунок 2.1 — Модель графа залежностей ресурсів

Управління залежностями в Terraform (рисунок 2.2) використовує граф залежностей для визначення порядку створення, оновлення та видалення ресурсів, спираючись на їх взаємні залежності. Цей підхід сприяє більш ефективному та безпомилковому управлінню інфраструктурою, забезпечуючи автоматизацію процесів та зменшення ризику помилок, особливо у складних системах [13].

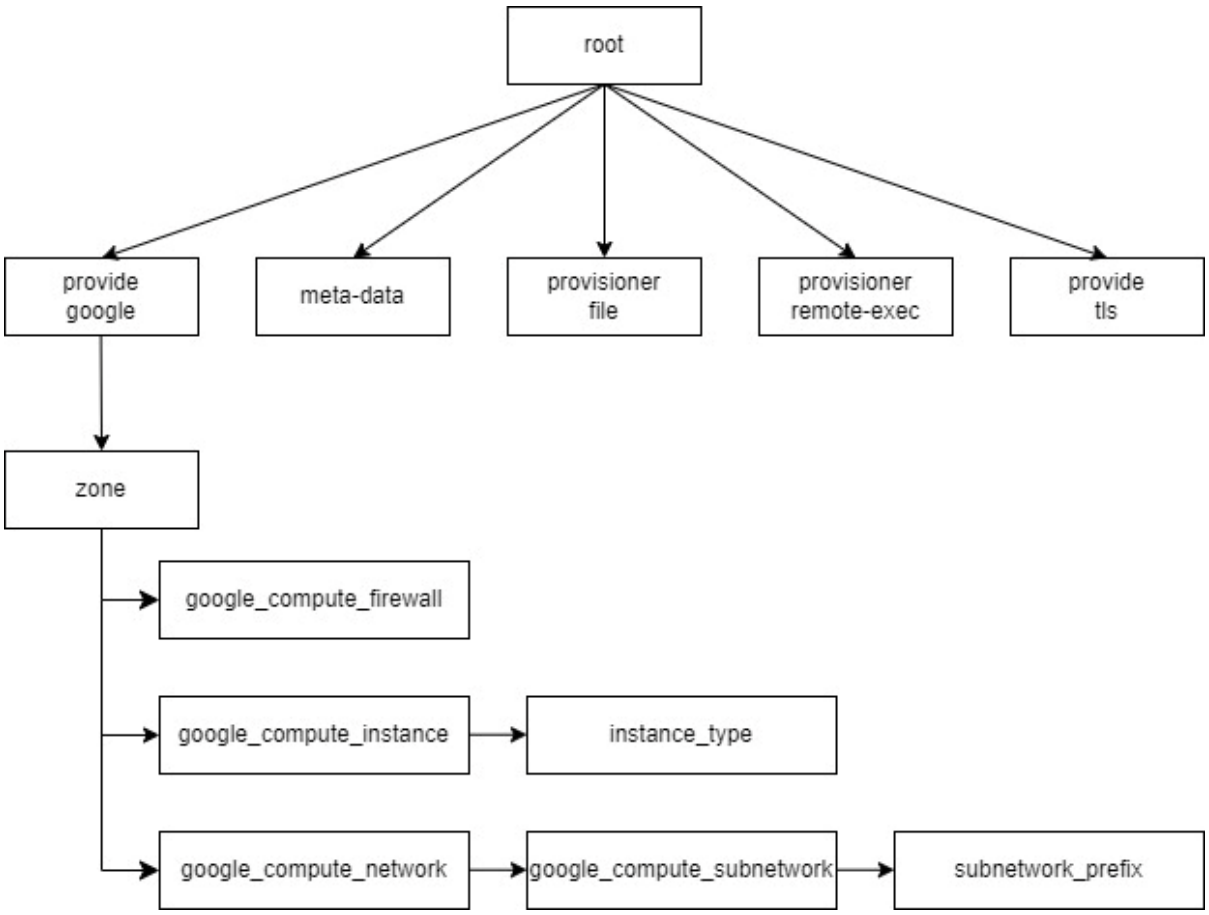

Рисунок 2.2 — Модель залежностей для ресурсу віртуальний сервер (instance)

2.2 Моделювання життєвого циклу інфраструктури

Інфраструктура може переходити через багато станів свого життєвого циклу. Коли ми створюємо нову інфраструктуру, клаудний провайдер виділяє для запуску виділяє ті чи інші ресурси (сервери, лоад-балансери чи віртуальні мережі). Далі, інфраструктура переходить у стадію підготовки, де вона готується до першого запуску. Протягом свого життя інфраструктура може бути неодноразово зупинена, перезапущена, оновлена, це продемонстровано на рисунок 2.3.

Проаналізувавши це можемо чітко визначити всі стани інфраструктури в котрих вона може перебувати:

— ПРОВІЗІЯ (PROVISIONING) — ресурси виділяються провайдером, але ще не запущені;

— ПІДГОТОВКА (**STAGING**) — ресурси отримані, готуються до першого запуску;

— ПРАЦЮЄ (**RUNNING**) — ресурс завантажується або працює;

— ЗУПИНЕННЯ (**STOPPING**) — ресурс зупиняється, ви запросили зупинку, або сталася помилка, це тимчасовий статус, після якого ресурс переходить у стан ЗАВЕРШЕНО (TERMINATED);

— РЕМОНТ (**REPAIRING**) — ресурс ремонтується, ремонт відбувається, коли ресурс зіткнулася з внутрішньою помилкою або через технічне обслуговування на стороні провайдера;

— ЗАВЕРШЕНО (**TERMINATED**) — ресурс зупинений, та може бути перезапущеним або видалити;

— ПРИЗУПИНЕННЯ (**SUSPENDING**) — ресурс знаходиться в процесі призупинення;

— ПРИЗУПИНЕНО (**SUSPENDED**) — ресурс перебуває в призупиненому стані, ви можете відновлювати або видалити його.

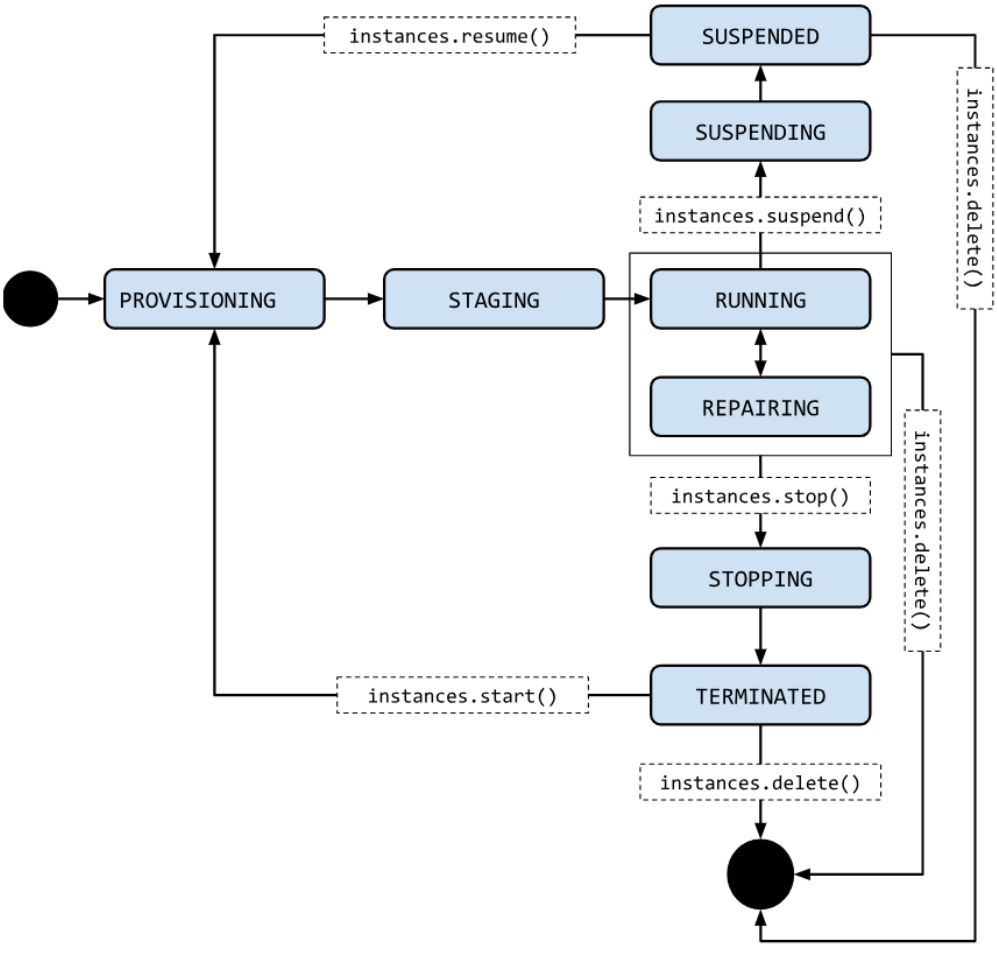

Рисунок 2.3 — Життєвий цикл інфраструктури

Отже, опис станів інфраструктури дає нам розуміння як цими процесами можна керувати і які переваги вони нам надають [13].

2.3 Математичні розрахунки по створенню інфраструктури

Розглянемо найпростішу інфраструктуру наведену у таблиці 2.1, її було створено за допомогою Web інтерфейсу, та зафіксовано приблизний час на створення та налаштування кожного ресурсу.

Таблиця 2.1 — Найпростіша хмарна інфраструктура

Назва ресурсу	Кількість (шт)	Час (хв)/одиницю
Server (сервер)	10	15
Network (мережа)	1	15
Sub-network (під-мережа)	2	15
Firewall (брадмаузер)	1	15
Load balancing (Балансування)	1	30
NAT	1	30

Для розрахунку часу, необхідного на створення 10 серверів, потрібно врахувати час ініціалізації кожного сервера та можливі затримки у мережі. Якщо нам відомо середній час створення одного сервера наведеного у таблиці 2.1, ми можемо просто помножити цей час на 10, враховуючи що сервери створюються послідовно через Web інтерфейсу [13,14].

$$t_{ser} = t_{create} * n_{server} \ , \qquad (2.1)$$

$$t_{ser} = 15 * 10 = 150 \ / 60 = \ 2{,}5 \text{ год.}$$

Якщо промоделювати створення мережі, то це залежить від складності її конфігурації, включаючи визначення діапазонів IP-адрес, маршрутизації та правил безпеки, але ми моделюємо найпростішу мережу. Для створення підмережі потрібно планування розподілу IP-адрес, встановлення маски підмережі та

інтеграції з основною мережею, що може зайняти додатковий час. Отже, використавши формулу, зробимо розрахунки:

$$t_{net} = t_{create} * n_{net.sub} , \quad (2.2)$$

$$t_{net} = 15 * 2 = 30 / 60 = 0{,}5 \text{ год.}$$

Для налаштування брандмауера через UI, залежить від складності його правил та політик безпеки, які можуть включати фільтрацію трафіку, блокування портів та інші параметри безпеки. Конфігурація брандмауера в складній мережевій інфраструктурі, особливо з багатьма підмережами та різними рівнями доступу, може вимагати додаткового часу для планування та тестування, ми ж моделюємо найпростіший брадмаузер.

$$t_{fw} = t_{create} * n_{fw} , \quad (2.3)$$

$$t_{fw} = 15 * 2 = 30 / 60 = 0{,}5 \text{ год.}$$

Налаштування NAT, залежить від складності мережевої архітектури та вимог до маршрутизації, включаючи кількість підмереж, які потребують трансляції адрес. Конфігурація NAT великої мережі з багатьма внутрішніми IP-адресами, які потребують доступу до зовнішнього Інтернету, може бути часозатратною через необхідність детального планування та тестування правил трансляції.

$$t_{nat} = t_{create} * n_{nat} , \quad (2.4)$$

$$t_{nat} = 30 * 2 = 60 / 60 = 1 \text{ год.}$$

Load balancing його налаштування, залежить від складності мережевої інфраструктури та вимог до розподілу трафіку, включаючи кількість серверів та сервісів, які потрібно балансувати. Конфігурація балансувальника навантаження в складних системах, які включають багатошарову архітектуру та різні типи трафіку (HTTP, HTTPS, TCP), може бути часозатратною через необхідність детального планування та тестування.

$$t_{lb} = t_{create} * n_{lb}, \quad (2.5)$$

$$t_{lb} = 30 * 2 = 60 / 60 = 1 \text{ год.}$$

Знаючи час, необхідний для створення основних ресурсів, таких як мережі, підмережі, брандмауери, NAT та балансувальники навантаження, сервери можна розрахувати загальний час $t_{general}$, потрібний для розгортання повноцінної мережевої інфраструктури. Це дозволяє ефективно планувати проекти та ресурси, забезпечуючи оптимізацію часу та зусиль, необхідних для реалізації мережевих рішень.

$$t_{general} = t_{ser} + t_{net} + t_{fw} + t_{nat} + t_{lb} + tn, \quad (2.6)$$

$$t_{general} = 2,5 + 0,5 + 0,5 + 1 + 1 = 5,5 \text{ год.}$$

Загальний час, необхідний для розгортання мережевої інфраструктури, складе 5,5 години, що дозволяє точно спланувати робочий процес та ресурси. Під час використання послуг хмарного провайдера, важливо враховувати, що за цей час буде нараховуватися плата за використані ресурси. Тому ефективне управління часом та ресурсами є ключовим для мінімізації витрат і оптимізації бюджету проекту [14].

Ми можемо визначити зробити розрахунки будь-якого члена послідовності: 0,25; 0,5; 0,75; 1; 1,25; 1,5; 1,75; 2; 2,25; 2,5, має загальну різницю d = 0,25, кожний наступний член на 0,25 більший за попередній, це демонструє рисунок 2.4.

Проведення математичних розрахунків часу, необхідного для створення 10 серверів, це дозволяє точно оцінити загальний час, потрібний для реалізації цього проекту.

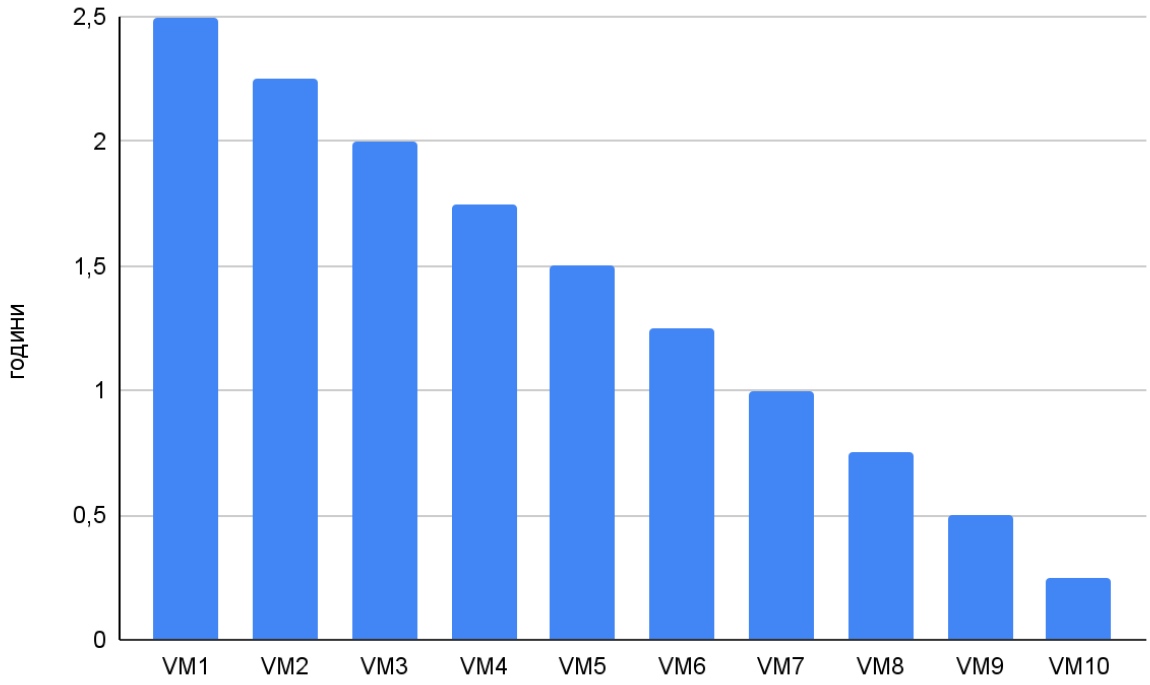

Рисунок 2.4 — Час простоювання кожного із серверів

Для визначення загальної суми арифметичної послідовності, яка складається з чисел 0,25; 0,5; 0,75; 1; 1,25; 1,5; 1,75; 2; 2,25; 2,5, можна застосувати формулу 2.7.

$$S_n = \frac{n}{2}(a + a_n), \tag{2.7}$$

де S_n — сума перших n членів;

a — перший член;

a_n — n-й член послідовності;

a = 0,25 (перший член);

a_n = 2,5 (десятий, тобто останній член);

n = 10 (кількість членів у послідовності).

$$S_{10} = \frac{10}{2}(0{,}25 + 2{,}5) = 5 * 2{,}75 = 13{,}75 \text{ год.}$$

Сумарний час роботи всіх серверів, які були створені між першим та останнім, та працюють відповідно до послідовності часу (0,25; 0,5; 0,75; 1; 1,25; 1,5; 1,75; 2; 2,25; 2,5 годин), складає 13,75 години.

Скориставшись формулою 2.8 для розрахунку вартості віртуальної машини $Price_{vm}$, де маємо загальний час, витрачений на створення або використання сервера, а також вартість однієї години використання. Цей підхід дозволяє точно визначити витрати на кожну віртуальну машину, виходячи з її часу використання та ставки оплати за годину [15].

Таке моделювання є корисним для бюджетування та планування витрат у проєкт, що включають використання серверів, особливо в хмарних середовищах, де вартість ресурсів часто розраховується на годинній основі.

Таблиця 2.2 — Білінг стандартного сервера

Machine type	Virtual CPUs	Memory	Price (USD)
e2-standard-2	2	8GB	0,086334
n2-standard-2	2	8GB	0.125124

В таблиці 2.2 показує розрахункову вартість для стандартних серверів у сімействі машин E2 та N2. Віртуальний процесор (Virtual CPUs) та пам'ять(Memory) кожного з цих типів машин розраховуються за їх індивідуальними попередньо визначеними цінами. В таблиці 2.2 показано вартість, яку ви можете очікувати, використовуючи конкретний тип машини [15].

$$Price_{vm} = t_{sum.vm} * Price, \quad (2.8)$$

$$Price_{vm} = 13,75 * 0.086334 = 1,18\ \$$$

За допомогою формули можна зробити розрахунок, що загальна вартість використання або створення серверів за вказаний часовий період становить 1,18 долара. Цей розрахунок враховує загальний час використання стандартного сервера, який складає 13,75 години, помножений на ставку ціни 0.086334 долара за годину [16].

Такий підхід дозволяє точно бюджетувати витрати на хмарні ресурси, однак він може не бути найбільш економічно вигідним, оскільки ручне створення та управління інфраструктурою часто вимагає значних фінансових витрат. Тому оптимізація процесу за допомогою технологія автоматизації розгортання

обчислювальної інфраструктури в хмарному середовищі може бути кращим рішенням для зниження витрат та підвищення ефективності використання хмарних ресурсів.

2.4 Групування інфраструктурних ресурсів

Модуль — це логічне об’єднання ресурсів, котрі можуть використовуватися у різних проєктах. Зазвичай ресурси модуля є ідентичними для різних проєктів, відрізняються тільки значення змінних [16].

Використовуючи модулі, ми уникаємо дублювання коду. Якщо нам треба створити 15 однакових серверів (які потребують security_group, security_group_rules, public_ip etc.) у різних проєктах, можемо написати один модуль і 15 разів його викликати.

$$N_{line} = N_{m1} + N_{m2} + N_{mn}\ , \tag{2.9}$$

Для розрахунку кількості стрічок коду для створення сервера bastion, для середовища DEV, можна визначити за формулою:

$$N_{DEV} = N_{server} + N_{network} + N_{fw} + N_{lb} + N_{nat}\ , \tag{2.10}$$

$$N_{DEV} = 15 + 8 + 12 + 20 + 20 = 75.$$

Наприклад коду обсягом близько 75 рядків, що включає створення ресурсів, такі як параметри безпеки, правила мережі та налаштування bastion хосту, забезпечує детальне управління та безпеку. Це також пришвидшує процес створення інфраструктур, як демонструється на рисунку 2.5.

Використання цього підходу в Terraform для модуля "bastion" сприяє ефективному розгортанню хмарних ресурсів. Це також дозволяє контрольоване управління ресурсами. Написаний модуль можна використовувати повторно, що збільшує його практичність [16].

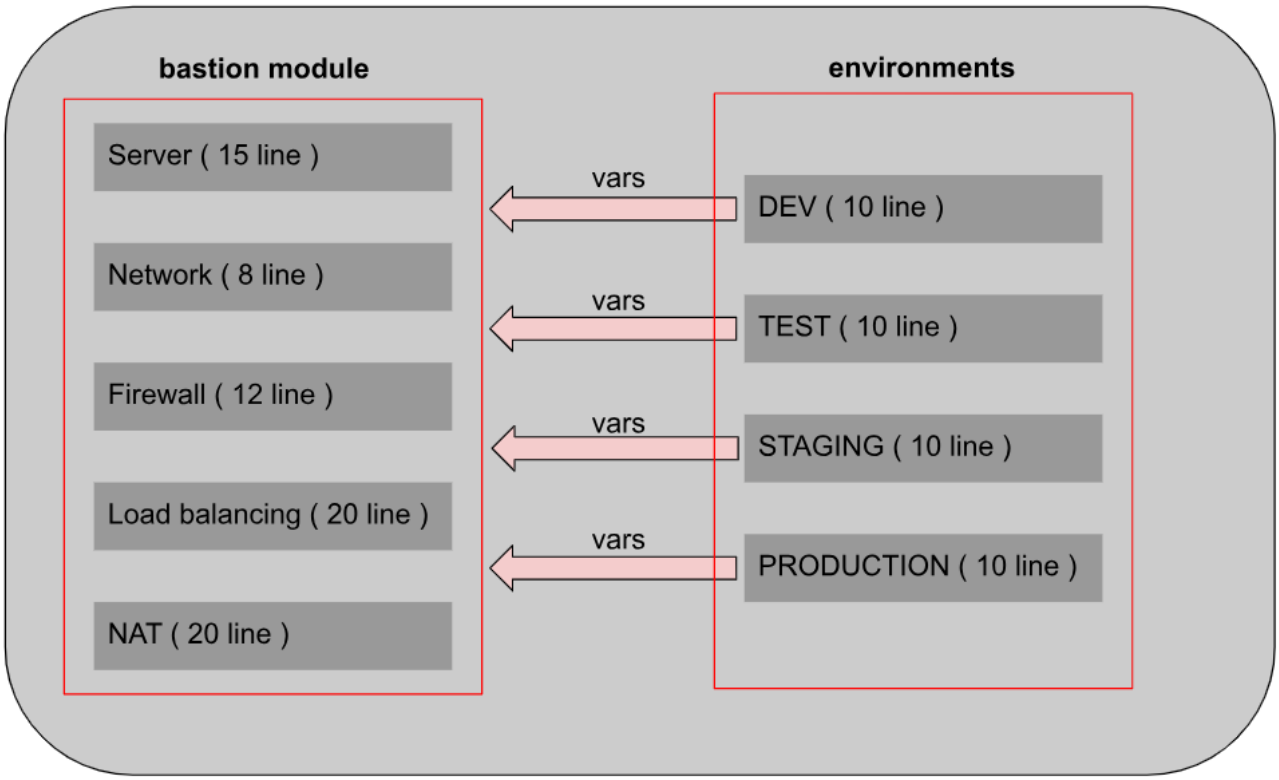

Рисунок 2.5 — Групування інфраструктурних ресурсів

Написання модулів спрощує процес розробки, дозволяючи повторно використовувати однакові ресурси в різних проектах. Єдина відмінність між проектами полягає в значеннях змінних, що робить модулі гнучкими і універсальними для різноманітних застосувань.

2.5 Алгоритм ефективного управління інфраструктурою

Алгоритми ефективного управління в хмарному обчисленні дозволяє оптимізувати розподіл та використання ресурсів, знижуючи витрати та підвищуючи продуктивність. Вона включає автоматизацію процесів, таких як масштабування, балансування навантаження та управління життєвим циклом ресурсів, щоб забезпечити ефективність та відповідність потребам користувачів. Завдяки алгоритмізації, хмарні системи можуть швидко адаптуватися до змін у навантаженні та потребах, забезпечуючи високий рівень доступності та надійності сервісів [16].

2.5.1 Алгоритм управління інфраструктурою, гілка Create

Для представлення процесу планування Terraform у вигляді алгоритму з використанням умовних операторів if-else.

Лістинг 1 — Алгоритм управління інфраструктурною, гілка Create

```
START
    Запустити Terraform
    IF файли конфігурації та стану існують
        THEN
        Прочитати файли конфігурації та стану
        Виконати розрахунки на основі прочитаної інформації
        IF потрібно створити ресурси
            THEN
            Виконати Create()
        ELSE IF потрібно оновити ресурси
            THEN
            Виконати Update()
        ELSE IF потрібно прочитати стан ресурсів
            THEN
            Виконати Read()
        ELSE IF потрібно видалити ресурси
            THEN
            Виконати Delete()
        ELSE
            Виконати No-op
            Згенерувати план виконання
            Вивести план виконання
        END
    ELSE
```

Вивести повідомлення про відсутність файлів конфігурації або стану

END

END

У графічному вигляді алгоритм матиме вигляд як показано на рисунку 2.6.

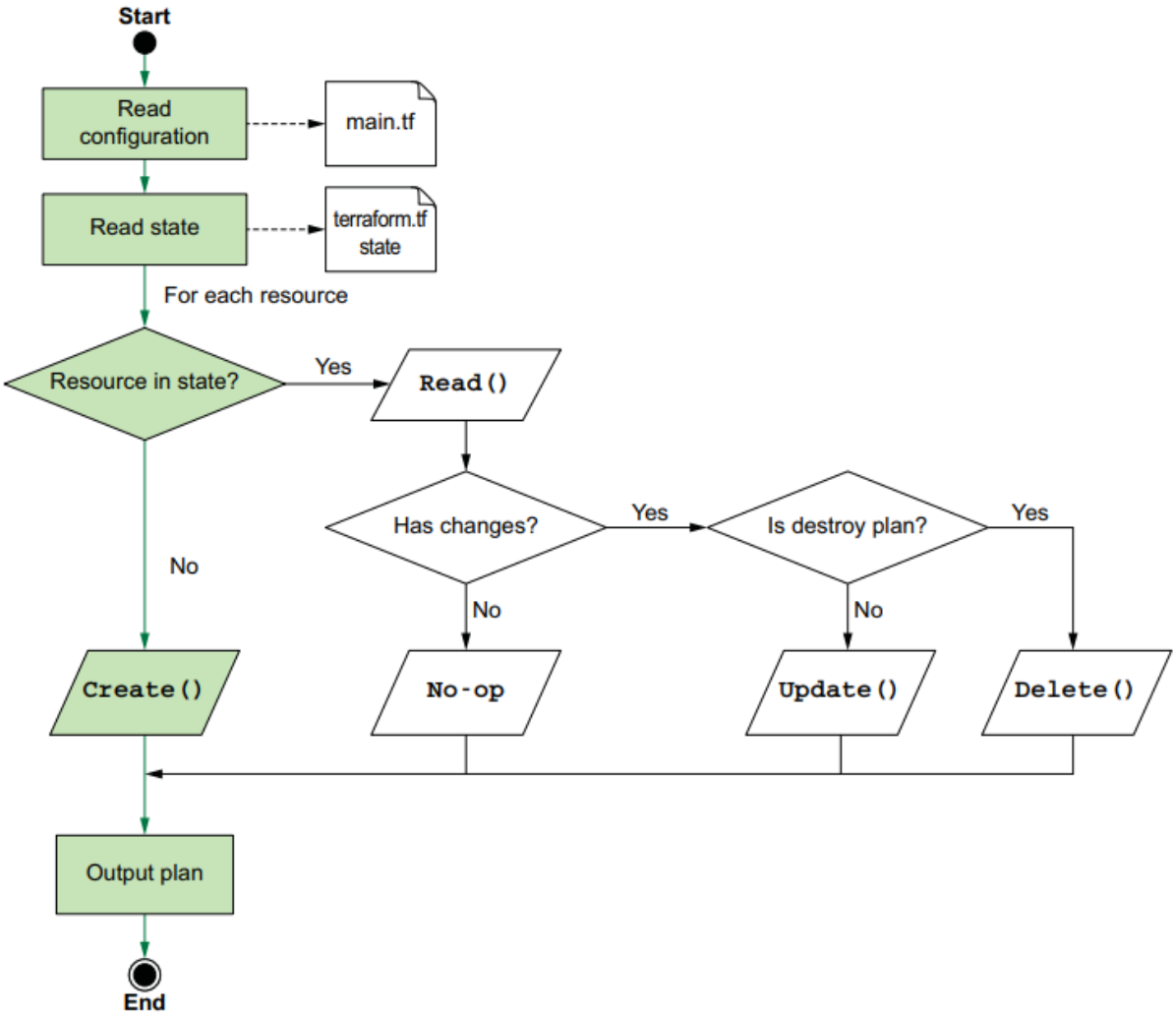

Рисунок 2.6 — Алгоритм управління інфраструктурою, гілка Create

Під час запуску Terraform виконує перший крок, де система аналізує конфігурацію, вичитуючи дані з конфігураційних файлів. Після цього, Terraform переходить до читання стану існуючої інфраструктури, перевіряючи наявність визначених ресурсів. У випадку, коли ресурс ще не існує, Terraform ініціює процес

його створення, забезпечуючи відповідність конфігурації та реального стану інфраструктури.

2.5.2 Алгоритм управління інфраструктурою, гілка Read

В наступному алгоритмі використовуються умовні оператори для перевірки наявності файлів стану та визначення дій, які потрібно виконати на основі результатів функції Read().

Лістинг 2 — Алгоритм управління інфраструктурою, гілка Read

```
START
    Запустити terraform plan
    IF файли стану існують
        THEN
        Для кожного ресурсу у файлі стану:
        Викликати Read()
        IF Read() не виявляє змін
        THEN
            Виконати операцію без дій (no-op)
        ELSE
            Виконати необхідні дії для досягнення бажаного стану
            Вивести результати Read()

        END
    END
END
```

У графічному вигляді алгоритм матиме вигляд як показано на рисунку 2.7.

При запуску Terraform, першим кроком є аналіз конфігураційних файлів, щоб зрозуміти заплановану структуру інфраструктури. Далі Terraform перевіряє стан існуючої інфраструктури, звіряючи його з конфігурацією. Якщо виявлено, що ресурс вже існує, результата виводить в консоль, що змін немає.

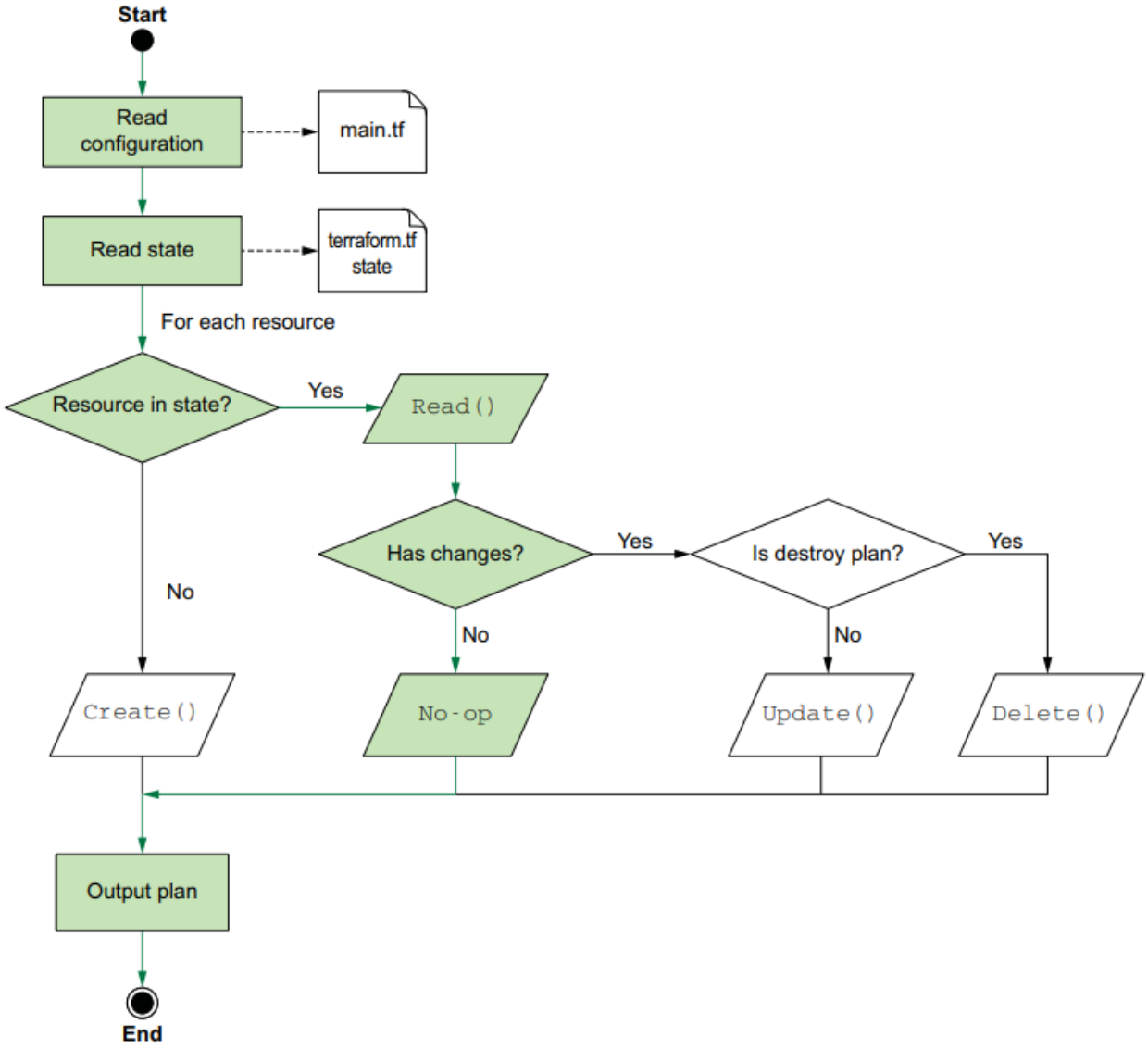

Рисунок 2.7 — Алгоритм управління інфраструктурою, гілка Read

2.5.3 Алгоритм управління інфраструктурою, гілка Update

Алгоритм процесу планування Terraform з використанням умовних операторів if-else для Update().

Лістинг 3 — Алгоритм управління інфраструктурою, гілка Update

START

Оновити код main.tf для відображення нового стану

Запустити terraform plan

IF terraform plan показує зміни

THEN

Вивести план виконання

IF змінено атрибут content

THEN

Terraform пропонує знищити старий ресурс і створити новий

IF content позначений як атрибут, що вимагає нового створення

THEN

Підготуватися до створення ресурсу заново

ELSE

Виконати оновлення атрибуту на місці

END

ELSE

Виконати інші необхідні оновлення Update()

END

Запустити terraform apply -auto-approve для застосування змін

Вивести повідомлення, що змін немає

END

END

У графічному вигляді алгоритм матиме вигляд як показано на рисунку 2.8.

Під час запуску Terraform відбувається аналіз конфігураційних файлів, метою якого є з'ясування планованої структури інфраструктури. Після цього Terraform здійснює перевірку поточного стану існуючої інфраструктури, порівнюючи її з налаштуваннями у конфігурації. У разі виявлення змін в існуючому ресурсі, Terraform автоматично оновлює його, замість створення нового, та відображає відповідне повідомлення про оновлення в консолі.

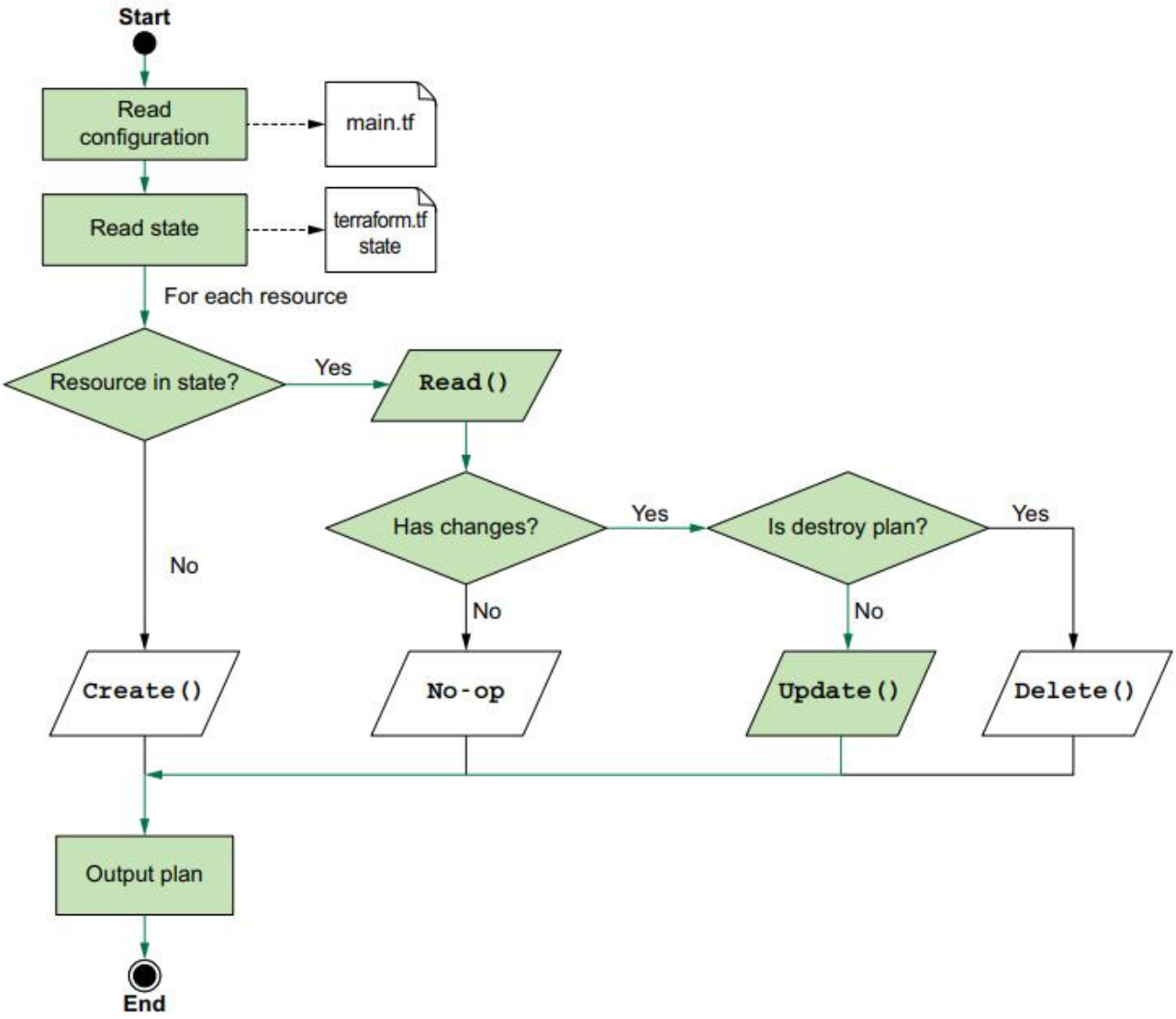

Рисунок 2.8 — Алгоритм управління інфраструктурою, гілка Update

2.5.4 Алгоритм управління інфраструктурою, гілка Delete

Алгоритм застосовує Delete() для того, щоб перевірити, чи є файли стану, та встановити, які кроки слід вжити для кожного ресурсу, виходячи з того, чи присутній він у файлі стану.

Лістинг 4 — Алгоритм управління інфраструктурною, гілка Delete

```
START
    Запустити terraform destroy -auto-approve
    IF файли стану існують
        THEN
```

```
                Для кожного ресурсу у файлі стану: Вивести результати Read()
                Викликати Read() для перевірки існування ресурсу
                IF ресурс існує
                        THEN
                        Викликати Delete() для видалення ресурсу
                        Вивести повідомлення про знищення ресурсу
                ELSE
                        Вивести повідомлення, що ресурс вже не існує
                END
        END
        Вивести повідомлення про завершення процесу знищення
        Вивести повідомлення про відсутність файлів стану
END
```

У графічному вигляді алгоритм матиме вигляд як показано на рисунку 2.9.

Початковим етапом при запуску Terraform є аналіз конфігураційних файлів, що дозволяє зрозуміти задуману структуру інфраструктури. На наступному кроці Terraform перевіряє відповідність між поточним станом інфраструктури та збереженим файлом стану. У разі виявлення розбіжностей, що свідчить про зміни або відсутність ресурсу в поточній конфігурації, Terraform видаляє такий ресурс, приводячи стан інфраструктури у відповідність з останньою конфігурацією.

Terraform використовує декларативний підхід для видалення ресурсів, де видалення здійснюється шляхом модифікації конфігураційних файлів, а не через окрему команду delete. Ресурси видаляються з конфігурації і підтверджуються через terraform apply, або можуть бути видалені з використанням команди terraform destroy, яка видаляє усі ресурси, описані в конфігурації. Для видалення конкретних ресурсів можна використовувати опцію -target з командою terraform destroy.

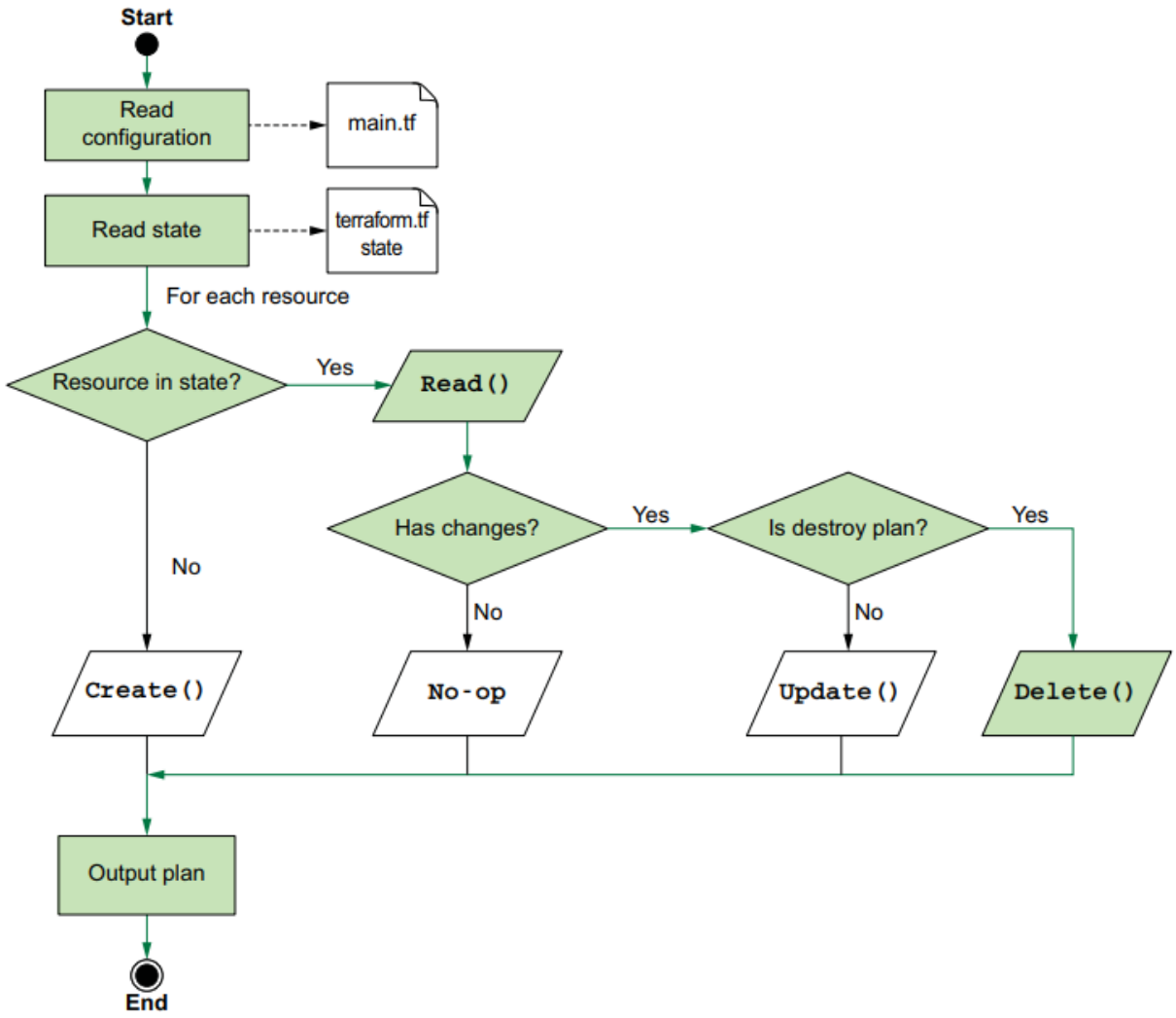

Рисунок 2.9 — Алгоритм управління інфраструктурною, гілка Delete

3 РОЗРОБКА IAC МОДУЛІВ РОЗГОРТАННЯ ІНФРАСТРУКТУРИ

Інструменти IaC передбачають те, що інфраструктура буде розгортатися у хмарному провайдері. Для виконання магістерської роботи був обраний провайдер Google Cloud Platform (GCP), як один з найпопулярніших та найбільш поширених провайдерів [16,17].

Перед початком роботи над модулями Terraform необхідно підготовити робочу машину до написання коду, встановити все необхідне та розробити структуру файлів конфігурації, а саме:

1) IDE для роботи; встановити IDE та розширення;
2) встановити Terraform;
3) створити обліковий запис GCP;
4) згенерувати GCP service account;
5) налаштування запити до GCP API;
6) розробка IaC за допомогою HCL;
   a) налаштування провайдера GCP;
   b) сворення мережі та підмережі;
   c) створення віртуальних серверів;
   d) створення NAT транслятора;
   e) створення балансування мережевого навантаження.

3.1 Налаштування середовища для розробки

Для виконання дипломної роботи буде використано Linux Ubuntu 22.04 LTS, отже всі наступні команди розглядатимуться в контексті Linux.

Інтегроване середовище розробки (скор. “IDE” від англ. “Integrated Development Environment”) — це програмне забезпечення, яке містить все необхідне для розробки, компіляції, та налагодження коду програм. Нам необхідно встановити одну з таких IDE для написання програм на мові HCL [16].

3.1.1 Встановлення IDE Visual Studio Code

Visual Studio Code (часто абревіатура VS Code) — це безкоштовне відкрите середовище розробки від Microsoft. Хоча це не є IDE, виключно призначеним для HCL, VS Code здатний використовуватися для розробки HCL завдяки своїм розширенням і гнучкості.

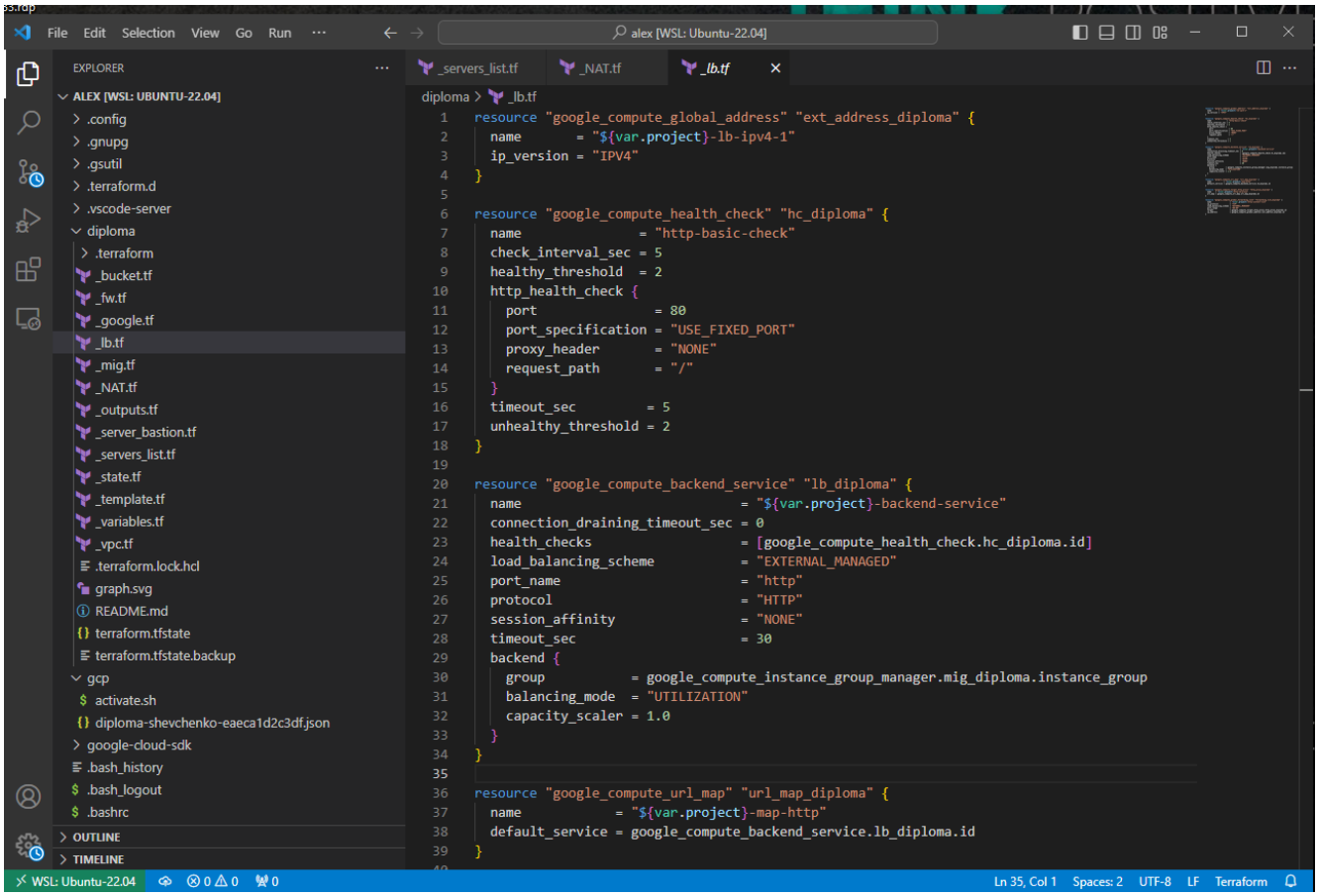

Рисунок 3.1 — Робоча область Visual Studio Code

Visual Studio Code (VS Code) вирізняється своєю потужністю та гнучкістю, роблячи його ідеальним інструментом для розробки на мові HCL та багатьох інших мовах. Інтелектуальне автодоповнення, форматування коду та вбудовані інструменти відлагодження роблять процес розробки на HCL більш продуктивним. Вбудована підтримка Git спрощує роботу з контролем версій, забезпечуючи комфортну інтеграцію з Git. Можливість налаштування та розширень робить VS Code універсальним редактором для розробки, незалежно від конкретних потреб користувача.

Розглянемо порядок виконання команд для встановлення VS Code:

1) запустіть «Термінал» через Ctrl + Alt + T або скористайтеся відповідною іконкою в меню;
2) введіть команду “sudo snap install --classic vscode” для завантаження та установки VS з офіційного сховища.

3.1.2 Встановлення розширень Terraform для IDE Visual Studio Code

Visual Studio Code (VS Code) включає базові функції, але за допомогою розширень можна значно розширити його функціонал, включаючи підтримку різних мов програмування та робочих процесів. Автори розширень можуть використовувати ті ж API, що і сам VS Code, для інтеграції з його інтерфейсом. Розширення можна переглядати та встановлювати через вкладку "Розширення" в VS Code. Для пошуку та встановлення специфічного розширення, наприклад "Terraform", достатньо скористатися функцією пошуку в цій вкладці. Розширення "HashiCorp Terraform" для VS Code забезпечує розширений функціонал для роботи з файлами Terraform, включаючи підсвічування синтаксису та інтелектуальні функції редагування [17].

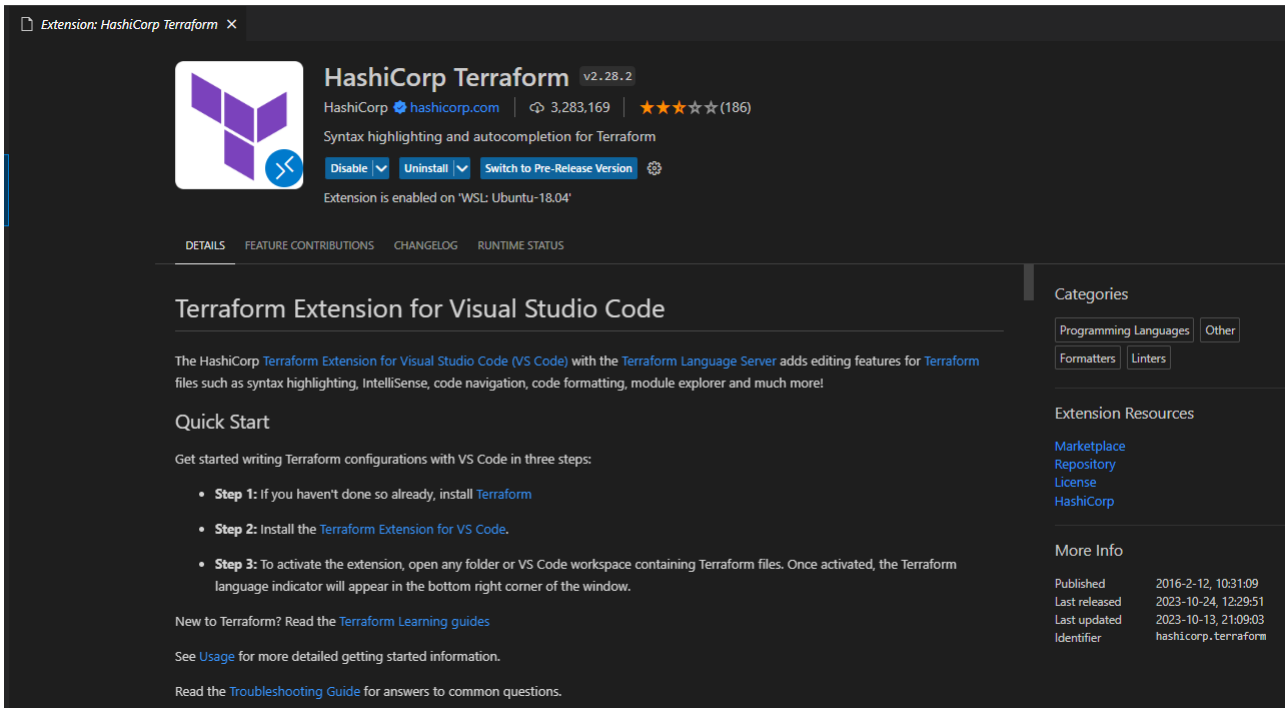

Рисунок 3.2 — Встановлення розширення Terraform

3.1.3 Встановлення Terraform

Виходячи з наявної ОС, нижче наведено інструкція для встановлення інструменту Terraform. Ми будемо встановлювати Terraform за допомогою пакетного менеджера apt-get [18].

Лістинг 3.1 — Встановлення Terraform

```
wget -O- https://apt.releases.hashicorp.com/gpg &&
sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg &&
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list &&
sudo apt update &&
sudo apt install terraform &&
terraform --version
```

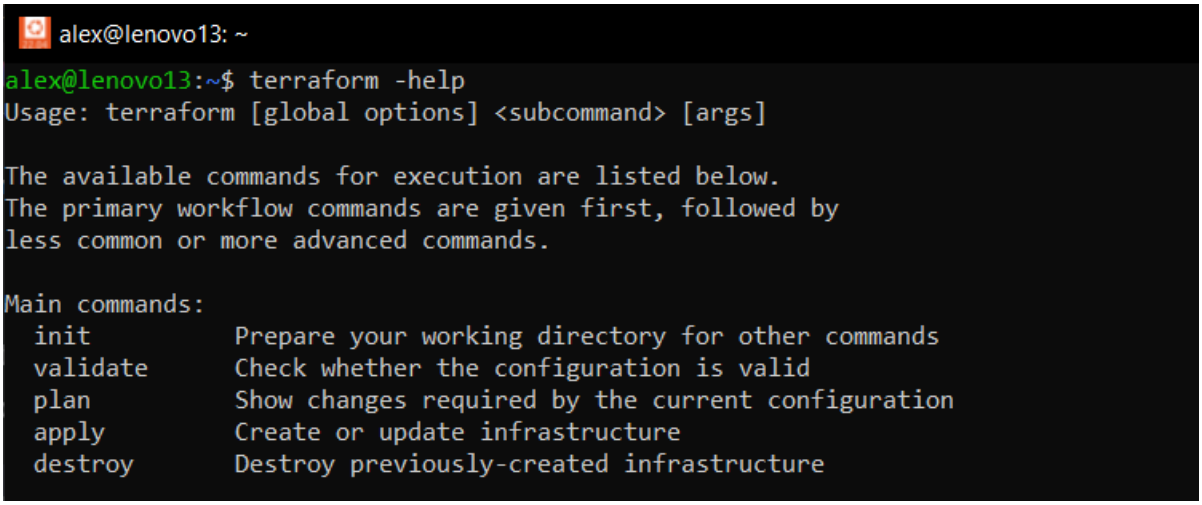

Рисунок 3.3 — Перевірка працездатності Terraform

Для керування модулями у Terraform є чотири основні команди:

— terraform init, підготувати робочий каталог для інших команд Terraform;

— terraform validate, перевіряє конфігураційні файли в робочому каталозі;

— terraform plan, переглянути зміни, для внести у вашу інфраструктуру;

— terraform apply, створення, оновлення або знищення інфраструктури;

— terraform destroy, знищення інфраструктури.

3.1.4 Встановлення Google Cloud SDK

Для роботи з хмарним провайдером Google Cloud Planform потрібно інсталювати на комп'ютер Google Cloud SDK.

Бібліотеки клієнтських SDK для популярних мов програмування, Cloud SDK надає специфічні для мови бібліотеки клієнтів Cloud, що підтримують природні конвенції та стилі кожної мови. Це спрощує взаємодію з API Google Cloud на мові котру ви вибрати, в нашому випадку це Hashicorp Configuration Language. Клієнтські бібліотеки також обробляють аутентифікацію, та асинхронного оброблення довготривалих операцій.

Лістинг 3.2 — Встановлення Google Cloud SDK

```
curl -O https://dl.google.com/dl/cloudsdk/channels/rapid/downloads/google-
cloud-sdk-453.0.0-linux-x86_64.tar.gz &&
tar zxvf google-cloud-sdk-453.0.0-linux-x86_64.tar.gz &&
./google-cloud-sdk/install.sh &&
./google-cloud-sdk/bin/gcloud init &&
gcloud components update
```

3.2 Створення облікового запису для GCP

Для того, щоб рухатися далі, згідно з поставленими задачами у даній дипломній роботі, необхідно створити GCP обліковий запис, в якому будуть виконуватися всі дії написані у коді Terraform.

Перейти за посиланням https://console.cloud.google.com, (рисунок 3.4.) де ми попадаємо на головну сторінку керування хмарними ресурсами, після чого потрібно активувати безкоштовний пробний період, натиснувши кнопку "START FREE" в правому верхньому куті. Далі буде декілька полів від хмарного провайдера, котрі потрібно заповнити, та активувати доступ до користування хмарними сервісами [19].

Працюючи з Google Cloud Platform(GCP) ми будемо оперувати таким терміном як “project”, (рисунок 3.5) проект це сукупність всіх наших ресурси в GCP, котрі складаються з налаштувань, дозволів та інших метаданих для того чи іншого ресурсу. Працюючи з GCP, ви використовуєте такі ідентифікатори, як ідентифікатор проекту (diploma-shevchenko) та номер проекту (diploma-shevchenko), у певних командах і викликах API.

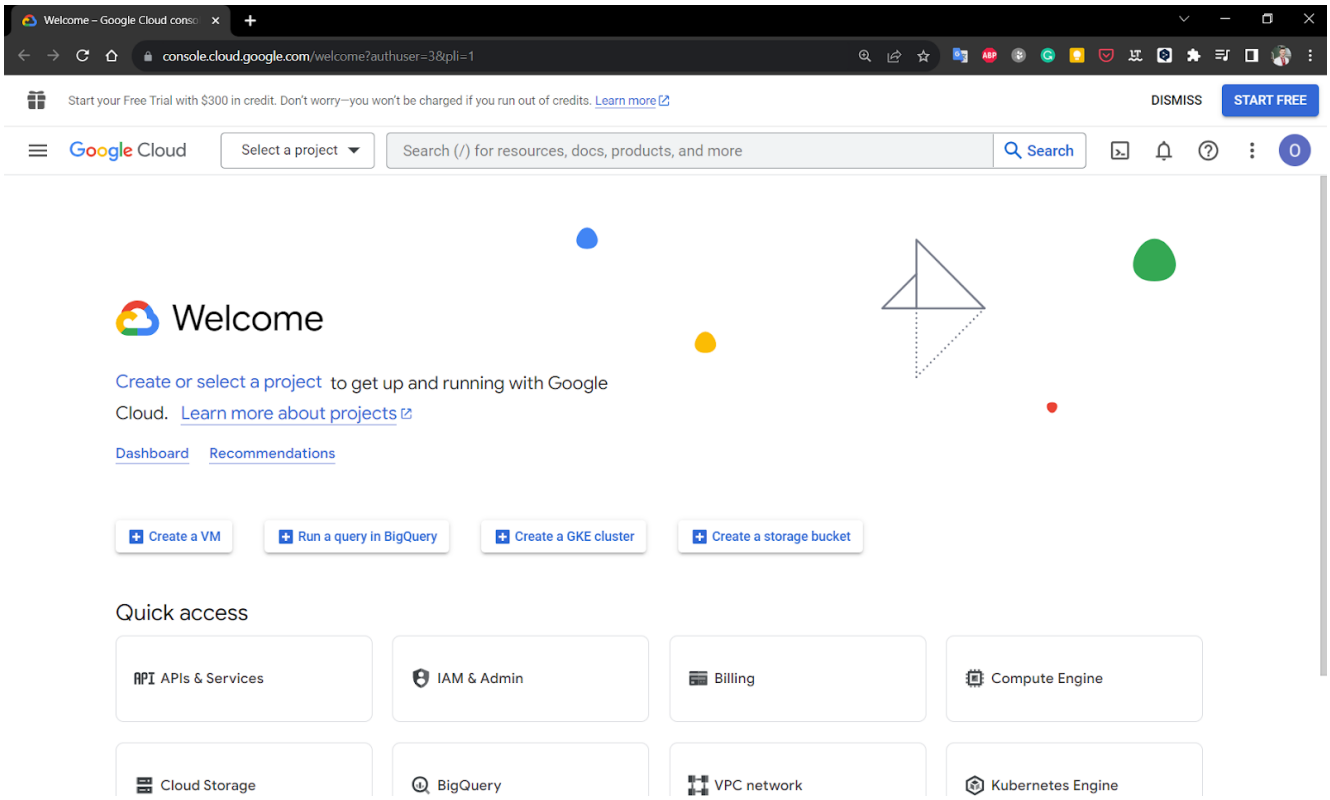

Рисунок 3.4 — Сторінка вітання Google Cloud Platform

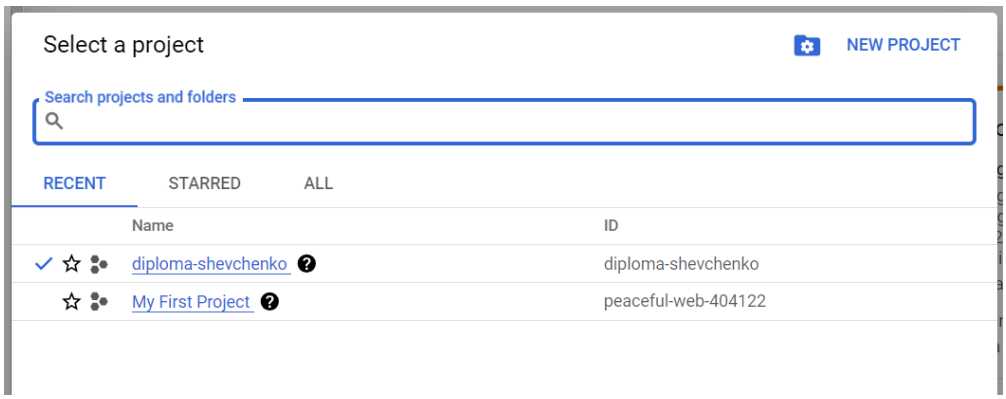

Рисунок 3.5 — Вибір ідентифікатора проекту (diploma-shevchenko)

Наступним кроком буде створення Service Account-a (SA). SA це спеціальний тип облікового запису в рамках проекту Google Cloud Platform (GCP), який використовується віртуальними машинами або програмою замість окремого кінцевого користувача.

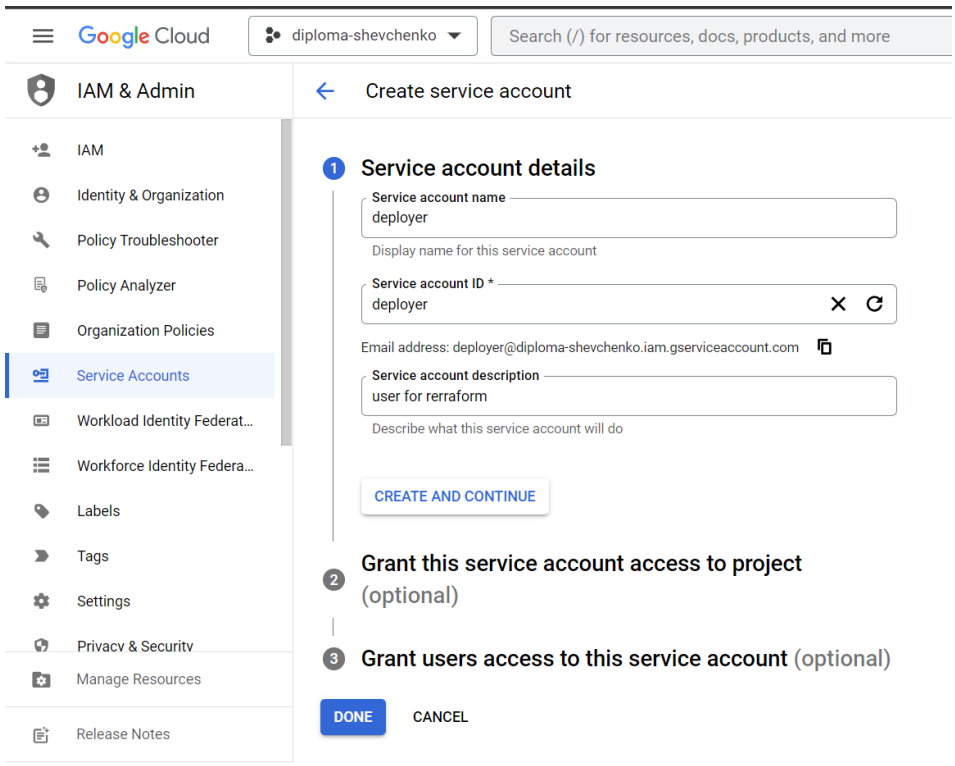

Рисунок 3.6 — Створення Service Account-a (SA)

Останнім кроком буде створення спеціального JSON-ключа, для доступу для SA до проекту, для цього тиснемо “Keys”, потім “Add key” і “Create new key”. Тип ключа потрібно вибрати “JSON” та натиснути кнопку “Create”, та зберегти ключ в надійному місці на файлові системі (рисунок 3.7).

Щоб зробити запити до GCP API, вам потрібно аутентифікуватися, щоб довести, що це ви робите запит. Бажаний метод розгортання ресурсів за допомогою Terraform на вашій робочій станції — використання Google Cloud SDK, про яке ми згадували раніше. Наразі єдині підтримувані облікові дані службового облікового запису — це облікові дані, завантажені з Cloud Console або створені за допомогою gcloud.

Ви надаєте ключ Terraform, використовуючи змінну оточення GOOGLE_APPLICATION_CREDENTIALS, встановлюючи значення на

розташування файлу за допомогою Command-line interface (CLI): export GOOGLE_APPLICATION_CREDENTIALS={{path}}, отже підставимо свої значення і отримаємо: export GOOGLE_APPLICATION_CREDENTIALS= =/home/alex/gcp/diploma-shevchenko-eaeca1d2c3df.json.

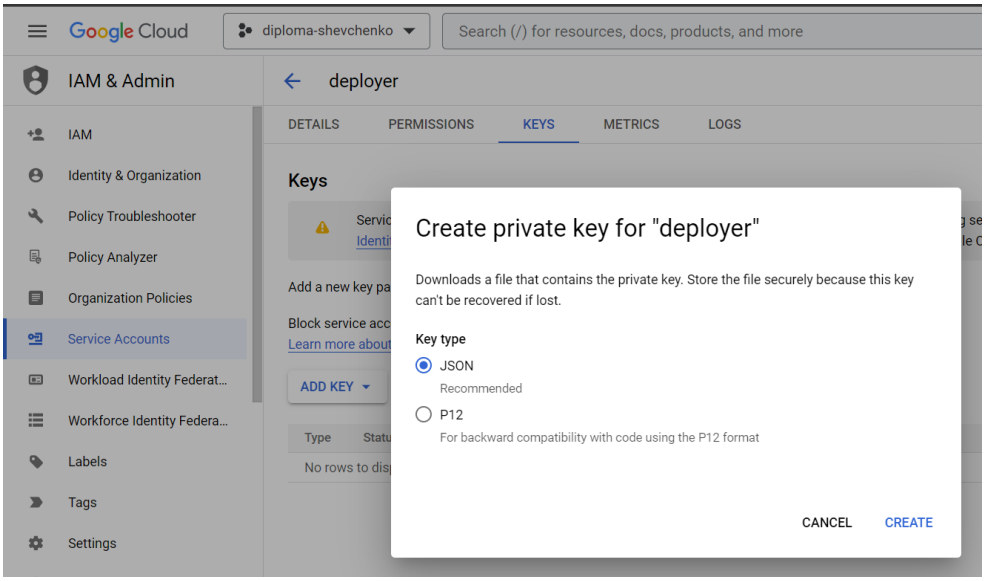

Рисунок 3.7 — Створення JSON-ключа доступу

```
alex@lenovo13:~/diploma$ gcloud services enable compute.googleapis.com
Operation "operations/acf.p2-917415936102-d832bdfb-7ed9-4937-85c4-b4c5997a71a9" finished successfully.
alex@lenovo13:~/diploma$
```

Рисунок 3.8 — Надання доступу до Google Cloud Platform API

Якщо потрібен доступу до обчислювальних ресурсів у хмарі через API, необхідно увімкнути compute.googleapis.com для повноцінного використання можливостей GCP (рисунок 3.8), виконавши команду в Command-line interface (CLI): gcloud services enable compute.googleapis.com [20].

3.3 Розробка модулів Terraform

Terraform використовує декларативну мову конфігурації, HCL (HashiCorp Configuration Language), для управління інфраструктурою, де користувачі описують бажаний стан інфраструктури, а не детальні кроки для її створення. Це дозволяє зосередитися на "що" потрібно зробити, замість "як". Також, Terraform

дозволяє організувати код у модулі, які є контейнерами для ресурсів, що використовуються разом, сприяючи повторному використанню коду та керуванню складністю інфраструктури. Це полегшує масштабування та управління інфраструктурою, роблячи процес більш ефективним та організованим [21].

Кожен із цих файлів Terraform (.tf) має специфічну роль у визначенні і управлінні інфраструктурою. В нашому випадку ми будемо працювати з наступними файлами:

— файл _variables.tf: визначає змінні, які використовуються в інших файлах Terraform;

— файл _bucket.tf: управляє конфігурацією сховищ (buckets) у хмарному сервісі, наприклад, Google Cloud Storage;

— файл _google.tf: містить конфігурацію для Google Cloud Platform (GCP), включаючи права доступу, регіони та інші параметри;

— файл _fw.tf: управляє налаштуваннями мережевого брандмауера або правилами безпеки;

— файл _lb.tf: управляє балансувальниками навантаження (load balancers);

— файл _mig.tf: керує managed instance groups (MIG) для масштабування та управління групами серверів;

— файл _NAT.tf: визначає NAT-шлюзи або інші ресурси для маршрутизації трафіку;

— файл _outputs.tf: містить визначення виводів Terraform, які використовуються для отримання інформації про інфраструктуру після її розгортання;

— файл _server_bastion.tf: створює bastion host (або jump server) для безпечного доступу до приватних мереж або серверів у хмарі;

— файл _servers_list.tf: містить конфігурацію або список серверів;

— файл _state.tf: управляє конфігурацією для зберігання та управління станом Terraform;

— файл _template.tf: містить шаблони для генерації динамічного вмісту;
файл _vpc.tf: створює та управляє віртуальними приватними мережами (VPC);

— файл terraform.tfstate: файл стану, який містить поточний стан управління інфраструктурою;

— файл terraform.tfstate.backup: резервна копія файлу стану Terraform.

Файл _variables.tf визначає змінні для Terraform проекту. Ці змінні використовуються для параметризації конфігурації, що дозволяє легко змінювати ключові значення без необхідності зміни основного коду Terraform. Вони можуть бути використані в інших файлах конфігурації Terraform для вказівки різних налаштувань, таких як ID проекту, регіон, зона, специфічні значення та діапазони IP-адрес тощо.

Лістинг 3.3 — Змінні для Terraform проекту

```
variable "project" {
  description = "Project ID"
  default     = "diploma-shevchenko"
  type        = string
}
variable "region" {
  description = "Region"
  default     = "europe-west2"
  type        = string
}
variable "zone" {
  description = "Region"
  default     = "c"
  type        = string
}
variable "var_diploma" {
  description = "diploma-shevchenko"
  default     = "diploma"
  type        = string
```

```
}
variable "ip_cidr_range_net_a" {
  description = "IP range 10.10.1.XX"
  default     = "10.10.1.0/24"
}
variable "ip_cidr_range_net_b" {
  description = "IP range 10.10.2.XX"
  default     = "10.10.2.0/24"
}
variable "nat_ips_count" {
  description = "IPs amount for Cloud NAT"
  default     = "1"
}
```

Файл _google.tf є ключовим елементом у загальній структурі Terraform проекту, оскільки він встановлює основні параметри для роботи з хмарним провайдером. Без цього файлу Terraform не зможе правильно управляти ресурсами в GCP). Він визначає провайдера google, який є важливим компонентом для керування ресурсами GCP за допомогою Terraform [21].

Файл _google.tf має наступний перелік елементів:

— provider "google": цей блок визначає провайдера для Terraform, у цьому випадку Google Cloud Platform (GCP);

— project = var.project: вказує Terraform використовувати змінну project для визначення GCP проекту, з яким він буде взаємодіяти, ця змінна має бути визначена десь у коді або передана як вхідний параметр під час виконання Terraform;

— region = var.region:, цей рядок визначає регіон у GCP, який буде використовуватися, значення цієї змінної також має бути визначено в коді Terraform або передано як вхідний параметр.

Лістинг 3.4 — Визначання провайдера хмарних послуг

```
provider "google" {
```

```
  project = var.project
  region  = var.region
}
```

Далі розглянемо файл _bucket.tf, його код наведено в лістингу 3.2, він створює сховище для зберігання станів та інших даних, забезпечуючи версіювання для кращого управління змінами і данними, та виводить URL цього сховища для подальшого використання.

Лістинг 3.5 — Управляння конфігурацією сховищ

```
resource "google_storage_bucket" "diploma_bucket_state" {
 name          = "diploma-shevchenko-state"
 force_destroy = false
 location      = var.region
 storage_class = "STANDARD"
 versioning {
  enabled = true
 }
}
output "url_diploma_bucket_state" {
        description = "bucket for state"
        value       = google_storage_bucket.diploma_bucket_state.self_link
}
```

Файл _vpc.tf визначає конфігурацію для створення віртуальної приватної мережі (VPC) та двох підмереж у Google Cloud Platform. Цей файл відіграє ключову роль у визначенні мережевої інфраструктури проекту в GCP, дозволяючи детально контролювати мережеву структуру та забезпечуючи гнучкість у розмежуванні ресурсів і послуг. Він дає змогу точно вказувати параметри мережі та підмереж, що є важливим для ефективного управління VPC та застосування політик безпеки.

Має наступний перелік елементів:

— auto_create_subnetworks встановлено на "false", що означає, що підмережі будуть створюватися вручну.[21]

Лістинг 3.6 — Створення віртуальної приватної мережі (VPC)

```
# VPC
resource "google_compute_network" "vpc_diploma" {
  provider              = google
  project               = var.project
  name                  = "${var.project}"
  auto_create_subnetworks = "false"
}
# Subnet subnet_a
resource "google_compute_subnetwork" "subnet_a" {
  provider      = google
  project       = var.project
  name          = "${var.project}-subnet-a"
  region        = var.region
  network       = google_compute_network.vpc_diploma.name
  ip_cidr_range = var.ip_cidr_range_net_a
}
# Subnet subnet_b
resource "google_compute_subnetwork" "subnet_b" {
  provider      = google
  project       = var.project
  name          = "${var.project}-subnet-b"
  region        = var.region
  network       = google_compute_network.vpc_diploma.name
  ip_cidr_range = var.ip_cidr_range_net_b
}
```

Файл _servers_list.tf використовується для створення та конфігурації набору віртуальних машин у Google Cloud. Він дозволяє керувати важливими параметрами Virtual machine (VM), такими як тип машини, образ ОС, розмір та тип диска, а також мережеві налаштування, що робить його ключовою частиною інфраструктури, що розгортається. Має наступний перелік елементів:

— count: Визначає кількість VM, що створюватимуться. Значення 10 означає, що буде створено 10 екземплярів;

— name: встановлює імена VM з використанням шаблону, що включає назву проекту (змінна var.project) та індекс (від 0 до 9). Це забезпечує унікальність кожної VM;

— machine_type: визначає тип обладнання для кожної VM, у цьому випадку "e2-standard-2";

— zone: встановлює зону розміщення VM, використовуючи змінні var.region та var.zone;

— labels: використовуються для надання додаткової метаінформації VM, наприклад, компонент системи та назву;

— tags: використовуються для групування та визначення політик доступу для VM;

— initialize_params: визначає параметри диска для запуску VM;

— image: визначає образ ОС, у цьому випадку Ubuntu 20.04;

— size: визначає розмір диска (у ГБ);

— type: визначає тип диска, у цьому випадку SSD (pd-ssd);

— network: визначає, до якої мережі буде підключено VM, використовується назва мережі, створеної як ресурс google_compute_network;

— subnetwork: визначає, до якої підмережі буде підключено VM, використовується назва підмережі, створеної як ресурс google_compute_subnetwork.

Лістинг 3.7 — Створення набору віртуальних машин (серверів)

```
resource "google_compute_instance" "server" {
```

```
count        = 10
name         = "${var.project}-server-${count.index}"
machine_type = "e2-standard-2"
zone         = "${var.region}-${var.zone}"
labels       = {
  component      = "${var.project}-server"
  name           = "server"
}
tags         = ["${var.project}-server"]
boot_disk {
  initialize_params {
    image = "ubuntu-2004-focal-v20231101"
    size  = 50
    type  = "pd-ssd"
  }
}
network_interface {
  network    = google_compute_network.vpc_diploma.name
  subnetwork = google_compute_subnetwork.subnet_a.name}}
```

Файл _fw.tf визначає правила брандмауера в рамках Terraform проекту для Google Cloud Platform, забезпечує критично важливі функції безпеки, визначаючи, який трафік може входити та виходити з мережі. Його правила забезпечують, що важливі сервіси моніторингу мають доступ до ресурсів, а внутрішні ресурси можуть взаємодіяти безпечно. Використання Terraform для управління правилами брандмауера дозволяє забезпечити послідовність та автоматизацію у впровадженні політик безпеки [21].

Має наступний перелік елементів:

— allow_healthcheck, правило дозволяє трафік з певних IP-адрес (зазначені діапазони, такі як 35.191.0.0/16, 130.211.0.0/22, 209.85.152.0/22, 209.85.204.0/22, 169.254.169.254), для моніторингу здоров'я системи чи сервісів;

— allow_internal_access, правило встановлюється для дозволу внутрішнього трафіку в мережі, обмеженого діапазоном 10.0.0.0/8. Також дозволяє всі протоколи, що сприяє внутрішньому спілкуванню та взаємодії між службами;

— ext_eccess_bastion, створює правило для доступу до bastion host, дозволяючи SSH (tcp порт 22) з будь-якої точки (вказано "0.0.0.0/0"), що може бути використане для безпечного підключення до внутрішніх ресурсів;

— port_80, правило встановлює доступ до HTTP-трафіку (порт 80).

Лістинг 3.8 — Створення правил брандмауера

```
resource "google_compute_firewall" "allow_healthcheck" {
  provider = google
  name     = "allow-healthcheck"
  network  = google_compute_network.vpc_diploma.name
  allow {
    protocol = "all"
  }
 source_ranges = ["35.191.0.0/16", "130.211.0.0/22", "209.85.152.0/22",
"209.85.204.0/22", "169.254.169.254"]
}
resource "google_compute_firewall" "allow_internal_access" {
  provider = google
  name     = "allow-internal-access"
  network  = google_compute_network.vpc_diploma.name
  allow {
    protocol = "all"
  }
  source_ranges = ["10.0.0.0/8"]
```

```
}
resource "google_compute_firewall" "ext_eccess_bastion" {
 name       = "ext-eccess-bastion"
 network    = google_compute_network.vpc_diploma.name
 priority   = 1001
 allow {
  protocol  = "tcp"
  ports     = ["22"]
 }
  source_ranges = ["0.0.0.0/0"]
}
resource "google_compute_firewall" "port_80" {
 name          = "app-port-80"
 network       = google_compute_network.vpc_diploma.name
 allow {
  protocol    = "tcp"
  ports       = ["80"]
 }
 source_ranges = ["0.0.0.0/0"]
  target_tags   = ["port-80"]
}
```

Файл конфігурації _lb.tf налаштовує базовий HTTP-балансувальник навантаження в GCP, з перевіркою здоров'я, службою backend service, HTTP-проксі та глобальним правилом перенаправлення для маршрутизації трафіку.

Має наступний перелік елементів:

— глобальна адреса Google Compute (google_compute_global_address), налаштована на використання адреси IPv4, та використовується для створення статичної IP-адреси для балансувальника навантаження;

— перевірка здоров'я Google Compute (google_compute_health_check), використовує HTTP-перевірку здоров'я на порті 80 з фіксованою специфікацією порту, без проксі-заголовка по шляху запиту /. Цей ресурс використовується для моніторингу стану екземплярів у балансувальнику навантаження;

— служба backend service Google Compute (google_compute_backend_service), налаштована для зовнішнього управління балансуванням навантаження з протоколом HTTP на порту під назвою "http", використовує перевірку здоров'я, визначену раніше (google_compute_health_check.hc_diploma.id), включає конфігурацію backend service, пов'язану з групою екземплярів (google_compute_instance_group_manager.mig_diploma.instance_group), з режимом балансування;

— правило перенаправлення (google_compute_global_forwarding_rule) трафіку, використовує протокол TCP зі схемою зовнішнього управління балансуванням навантаження, налаштована для діапазону портів "80-80" (що вказує на порт 80).

Лістинг 3.9 — Налаштування базового HTTP-балансувальник навантаження

```
resource "google_compute_global_address" "ext_address_diploma" {
  name       = "${var.project}-lb-ipv4-1"
  ip_version = "IPV4"
}
resource "google_compute_health_check" "hc_diploma" {
  name               = "http-basic-check"
  check_interval_sec = 5
  healthy_threshold  = 2
  http_health_check {
    port               = 80
    port_specification = "USE_FIXED_PORT"
    proxy_header       = "NONE"
    request_path       = "/"
```

```
  }
  timeout_sec        = 5
  unhealthy_threshold = 2
}
resource "google_compute_backend_service" "lb_diploma" {
  name = "${var.project}-backend-service"
  connection_draining_timeout_sec = 0
        health_checks= [google_compute_health_check.hc_diploma.id]
  load_balancing_scheme = "EXTERNAL_MANAGED"
  port_name                  = "http"
  protocol                   = "HTTP"
  session_affinity            = "NONE"
  timeout_sec                = 30
  backend {
    group =
google_compute_instance_group_manager.mig_diploma.instance_group
    balancing_mode  = "UTILIZATION"
    capacity_scaler = 1.0
  }
}
resource "google_compute_url_map" "url_map_diploma" {
  name            = "${var.project}-map-http"
  default_service = google_compute_backend_service.lb_diploma.id
}
resource "google_compute_target_http_proxy" "http_proxy_diploma" {
   name    = "${var.project}-http-lb-proxy"
  url_map = google_compute_url_map.url_map_diploma.id
}
resource "google_compute_global_forwarding_rule" "forwarding_rule_diploma"
{
```

```
  name = "${var.project}-http-content-rule"
  ip_protocol          = "TCP"
  load_balancing_scheme = "EXTERNAL_MANAGED"
  port_range           = "80-80"
  target               = google_compute_target_http_proxy.http_proxy_diploma.id
    ip_address           = google_compute_global_address.ext_address_diploma.id
}
```

Файл _NAT.tf містить конфігурацію для створення ресурсів NAT (Network Address Translation) у Google Cloud Platform, використовуючи Terraform. Цей файл є важливим для забезпечення зв'язку між приватними ресурсами у VPC та зовнішнім Інтернетом, дозволяючи безпечний і контрольований доступ до Інтернету для машин без публічних IP-адрес. Використання NAT сприяє підвищенню безпеки, оскільки внутрішні адреси не відображаються безпосередньо на Інтернеті [22].

Має наступний перелік елементів:

— ресурс "google_compute_router", "router_diploma", встановлює маршрутизатор у визначеному регіоні (var.region) для вказаної мережі (google_compute_network.vpc_diploma.id);

— ресурс "google_compute_address" "addresses_diploma", створює одну або більше зовнішніх IP-адрес, які будуть використовуватися для NAT. Кількість IP-адрес визначається змінною var.nat_ips_count, а їх назви формуються з використанням шаблону, що включає назву проекту та індекс;

— ресурс "google_compute_router_nat" "nat_diploma", визначає конфігурацію NAT на маршрутизаторі, вказаному вище. Встановлює налаштування NAT, такі як ім'я NAT, асоційоване з маршрутизатором.

Лістинг 3.10 — Створення ресурсів NAT (Network Address Translation)

```
# Router
resource "google_compute_router" "router_diploma" {
  provider = google
```

```
  name     = "${var.project}-router"
  region   = var.region
  network  = google_compute_network.vpc_diploma.id
}
# NAT Addresses
resource "google_compute_address" "addresses_diploma" {
  count  = var.nat_ips_count
  name   = "${var.project}-nat-ip-${count.index}"
  region = var.region
}
# NAT
resource "google_compute_router_nat" "nat_diploma" {
  provider = google
  name = "${var.project}-router-nat"
  router = google_compute_router.router_diploma.name
  region                             = var.region
  nat_ip_allocate_option             = "MANUAL_ONLY"
  nat_ips                            =
google_compute_address.addresses_diploma.*.self_link
    source_subnetwork_ip_ranges_to_nat =
"ALL_SUBNETWORKS_ALL_IP_RANGES"
  min_ports_per_vm                   = 8000
  udp_idle_timeout_sec               = 100
  icmp_idle_timeout_sec              = 60
  tcp_established_idle_timeout_sec   = 90
  tcp_transitory_idle_timeout_sec    = 90
  tcp_time_wait_timeout_sec          = 90
  enable_endpoint_independent_mapping = false
  log_config {
    enable = true
```

```
    filter = "ERRORS_ONLY"
  }
}
```

4 ВЕРИФІКАЦІЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Розділ має на меті дослідити та оцінити ефективність використання Infrastructure as Code (IaC). У цьому розділі буде проведено ретельний аналіз процесів автоматизованого створення інфраструктури в хмарі.

Перейшовши в робочий каталог ~/diploma потрібно виконати terraform plan, він робить три речі:

— гарантує актуальність стану, читаючи поточний стан будь-якої вже існуючої віддаленої інфраструктури;

— визначає різницю між поточною конфігурацією та попередніми даними стану;

— пропонує серію змін, які забезпечать відповідність віддаленої інфраструктури поточній конфігурації.

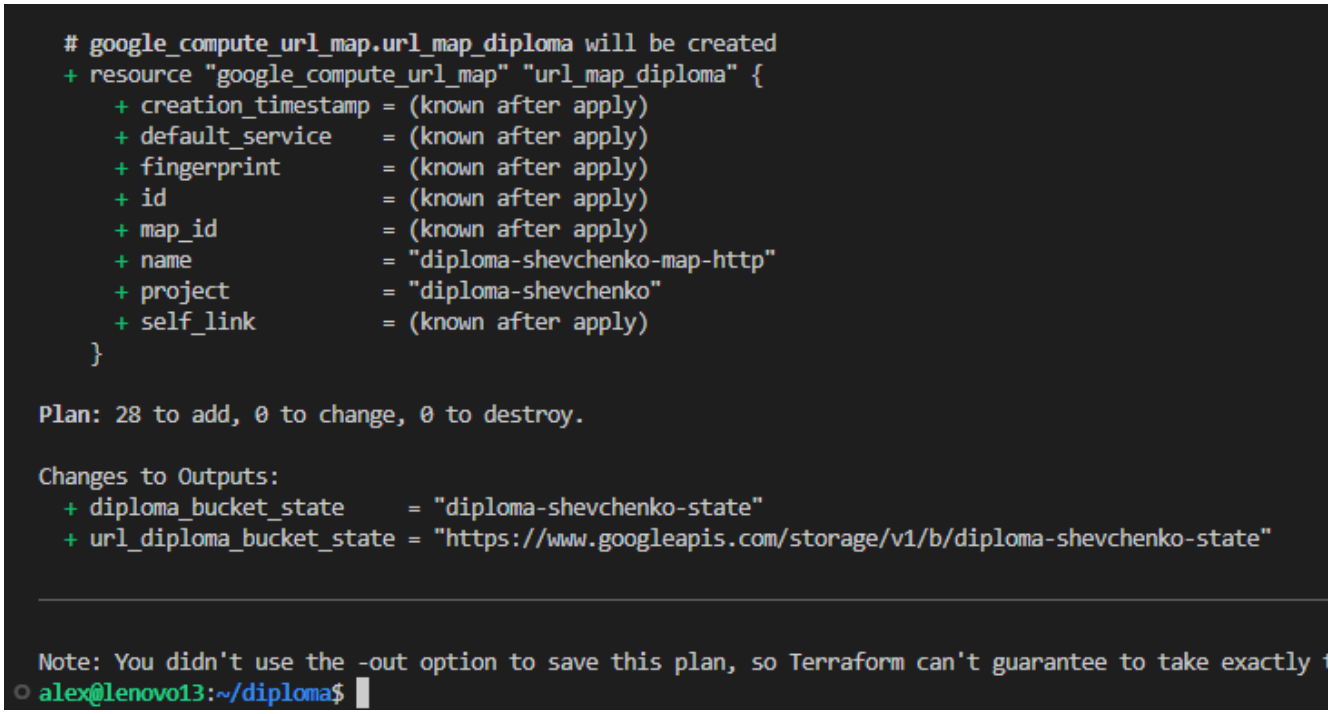

Рисунок 4.1 — Результат роботи команди terraform plan

Команда terraform apply у Terraform використовується для впровадження змін у інфраструктуру, які були визначені та заплановані командою terraform plan (рисунок 4.1).

Опис функціоналу terraform apply:

— виконання запланованих змін, коли ви виконуєте terraform apply, Terraform виконує зміни, які були ідентифіковані під час виконання terraform plan, ці зміни можуть включати створення, оновлення або видалення ресурсів у інфраструктурі;

— взаємодія з провайдерами, Terraform взаємодіє з провайдерами хмарних послуг або іншими сервісами, щоб впровадити зміни, Terraform для управління ресурсами GCP, terraform apply звернеться до GCP API для створення, оновлення або видалення ресурсів;

— оновлення стану, після виконання змін Terraform оновлює файл стану, цей файл відображає останній стан вашої інфраструктури та використовується Terraform для відстеження ресурсів та їх змін;

— звіт про виконані дії: по закінченню виконання команди, Terraform надає детальний звіт про виконані дії, зазначаючи, які ресурси були створені, оновлені або видалені;

— підтвердження перед застосуванням змін, зазвичай, перед виконанням terraform apply, Terraform вимагає підтвердження від користувача, щоб запобігти випадковим або неочікуваним змінам у інфраструктурі.

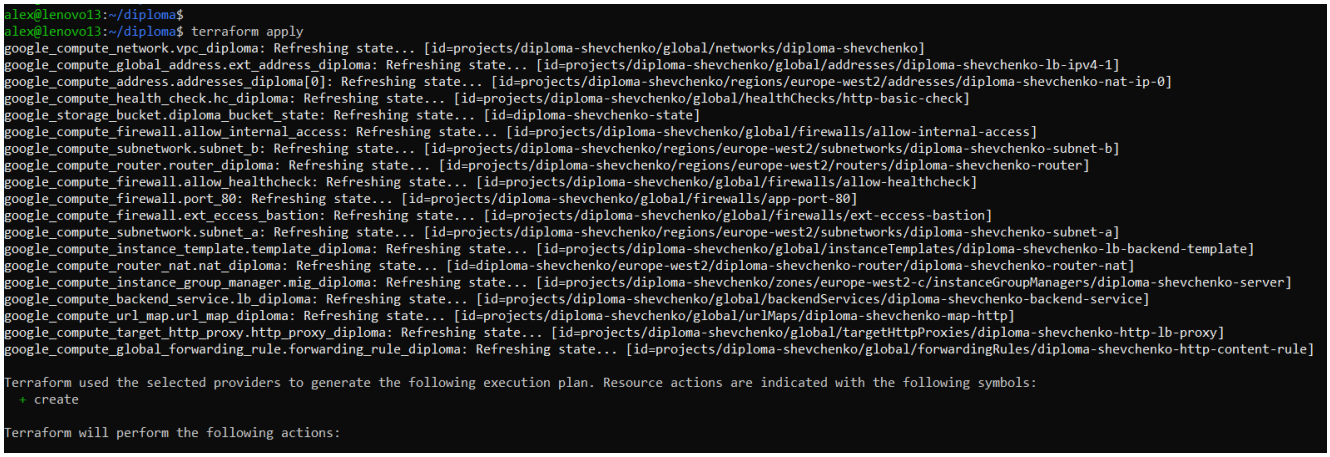

Рисунок 4.2 — Результат роботи команди terraform apply

4.1 Верифікація створення ресурсів

Ми будемо створювати лише основні ресурси наведені в порівняльні таблиці 4.1 для перевірки працездатності написаного коду, розглянемо:

— Network (мережа) та Sub-network (під-мережа), файл _vpc.tf;

— Server (сервер), файл _servers_list.tf;

— Firewall (брадмаузер), файл _fw.tf;

— NAT, файл _NAT.tf;

— Load balancing (Балансування), файли _lb.tf та _mig.tf

4.1.1 Верифікація Network

Перейшовши в робочий каталог ~/diploma виконаємо команду terraform apply. Результатом виконання в консолі відобразиться процес створення показано на рисунку 4.3 та рисунку 4.4.

```
Plan: 3 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  ~ diploma_vpc_name         = "projects/diploma-shevchenko/global/networks/diploma-shevchenko-diploma" -> (known after apply)

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

google_compute_network.vpc_diploma: Creating...
google_compute_network.vpc_diploma: Still creating... [10s elapsed]
google_compute_network.vpc_diploma: Still creating... [20s elapsed]
google_compute_network.vpc_diploma: Creation complete after 24s [id=projects/diploma-shevchenko/global/networks/diploma-shevchenko-diploma]
google_compute_subnetwork.subnet_b: Creating...
google_compute_subnetwork.subnet_a: Creating...
google_compute_subnetwork.subnet_b: Still creating... [10s elapsed]
google_compute_subnetwork.subnet_a: Still creating... [10s elapsed]
google_compute_subnetwork.subnet_a: Creation complete after 11s [id=projects/diploma-shevchenko/regions/europe-west2/subnetworks/diploma-shevchenko-subnet-a]
google_compute_subnetwork.subnet_b: Still creating... [20s elapsed]
google_compute_subnetwork.subnet_b: Creation complete after 22s [id=projects/diploma-shevchenko/regions/europe-west2/subnetworks/diploma-shevchenko-subnet-b]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

diploma_bucket_state = "diploma-shevchenko-state"
diploma_subnet_a = "diploma-shevchenko-subnet-a"
diploma_subnet_b = "diploma-shevchenko-subnet-b"
diploma_vpc_name = "projects/diploma-shevchenko/global/networks/diploma-shevchenko-diploma"
url_diploma_bucket_state = "https://www.googleapis.com/storage/v1/b/diploma-shevchenko-state"
alex@lenovo13:~/diploma$
```

Рисунок 4.3 — Результат верифікація Network в консолі

Відповідні зміни можна побачити в Web-інтерфейсі.

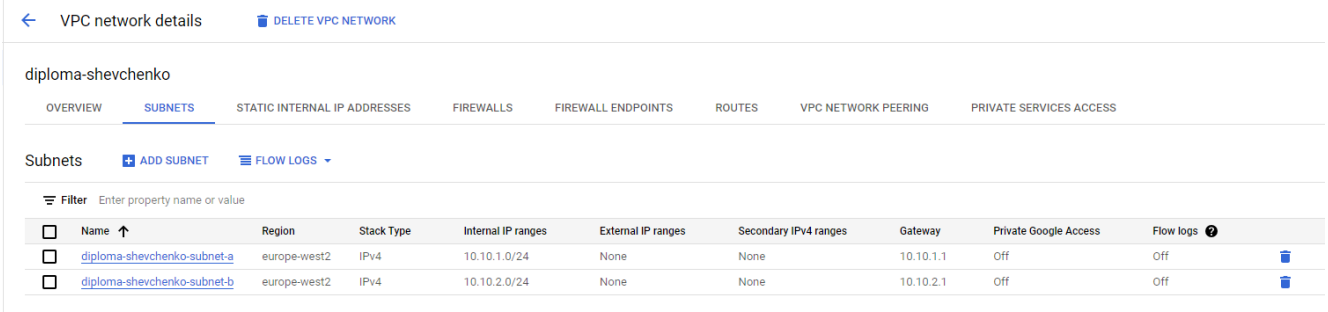

Рисунок 4.4 — Результат верифікація Network в Web-інтерфейсі

4.1.2 Верифікація Servers

Перемістившись у робочий каталог ~/diploma та виконавши команду terraform apply, ми побачимо в консолі процес створення як результат виконання (рисунок 4.5).

```
  Enter a value: yes

google_compute_instance.server[8]: Creating...
google_compute_instance.server[3]: Creating...
google_compute_instance.server[1]: Creating...
google_compute_instance.server[0]: Creating...
google_compute_instance.server[2]: Creating...
google_compute_instance.server[9]: Creating...
google_compute_instance.server[7]: Creating...
google_compute_instance.server[5]: Creating...
google_compute_instance.server[4]: Creating...
google_compute_instance.server[6]: Creating...
google_compute_instance.server[8]: Still creating... [10s elapsed]
google_compute_instance.server[3]: Still creating... [10s elapsed]
google_compute_instance.server[1]: Still creating... [10s elapsed]
google_compute_instance.server[0]: Still creating... [10s elapsed]
google_compute_instance.server[2]: Still creating... [10s elapsed]
google_compute_instance.server[9]: Still creating... [10s elapsed]
google_compute_instance.server[7]: Still creating... [10s elapsed]
google_compute_instance.server[5]: Still creating... [10s elapsed]
google_compute_instance.server[4]: Still creating... [10s elapsed]
google_compute_instance.server[6]: Still creating... [10s elapsed]
google_compute_instance.server[8]: Still creating... [20s elapsed]
google_compute_instance.server[3]: Still creating... [20s elapsed]
google_compute_instance.server[1]: Still creating... [20s elapsed]
google_compute_instance.server[0]: Still creating... [20s elapsed]
google_compute_instance.server[2]: Still creating... [20s elapsed]
google_compute_instance.server[9]: Still creating... [20s elapsed]
google_compute_instance.server[7]: Still creating... [20s elapsed]
google_compute_instance.server[5]: Still creating... [20s elapsed]
google_compute_instance.server[4]: Still creating... [20s elapsed]
google_compute_instance.server[6]: Still creating... [20s elapsed]
google_compute_instance.server[0]: Creation complete after 23s [id=projects/diploma-shevchenko/zones/europe-west2-c/instances/diploma-shevchenko-server-0]
google_compute_instance.server[2]: Creation complete after 24s [id=projects/diploma-shevchenko/zones/europe-west2-c/instances/diploma-shevchenko-server-2]
google_compute_instance.server[9]: Creation complete after 24s [id=projects/diploma-shevchenko/zones/europe-west2-c/instances/diploma-shevchenko-server-9]
google_compute_instance.server[5]: Creation complete after 24s [id=projects/diploma-shevchenko/zones/europe-west2-c/instances/diploma-shevchenko-server-5]
google_compute_instance.server[8]: Creation complete after 25s [id=projects/diploma-shevchenko/zones/europe-west2-c/instances/diploma-shevchenko-server-8]
google_compute_instance.server[3]: Creation complete after 24s [id=projects/diploma-shevchenko/zones/europe-west2-c/instances/diploma-shevchenko-server-3]
google_compute_instance.server[7]: Creation complete after 24s [id=projects/diploma-shevchenko/zones/europe-west2-c/instances/diploma-shevchenko-server-7]
google_compute_instance.server[1]: Creation complete after 25s [id=projects/diploma-shevchenko/zones/europe-west2-c/instances/diploma-shevchenko-server-1]
google_compute_instance.server[6]: Creation complete after 25s [id=projects/diploma-shevchenko/zones/europe-west2-c/instances/diploma-shevchenko-server-6]
google_compute_instance.server[4]: Creation complete after 25s [id=projects/diploma-shevchenko/zones/europe-west2-c/instances/diploma-shevchenko-server-4]

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.

Outputs:

diploma_bucket_state = "diploma-shevchenko-state"
diploma_subnet_a = "diploma-shevchenko-subnet-a"
diploma_subnet_b = "diploma-shevchenko-subnet-b"
diploma_vpc_name = "projects/diploma-shevchenko/global/networks/diploma-shevchenko"
url_diploma_bucket_state = "https://www.googleapis.com/storage/v1/b/diploma-shevchenko-state"
alex@lenovo13:~/diploma$
```

Рисунок 4.5 — Результат верифікація Server в консолі

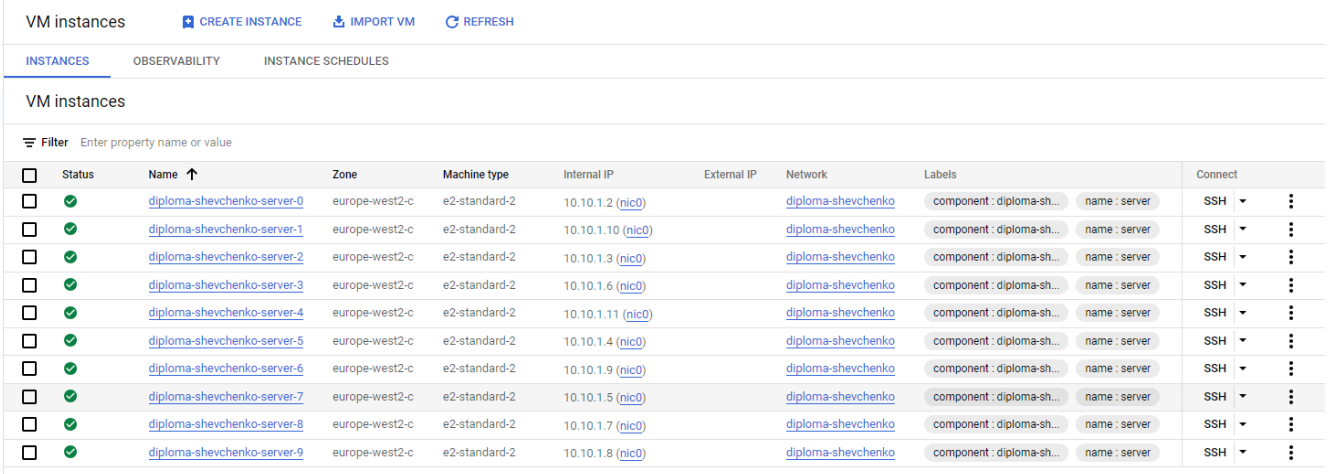

Рисунок 4.6 — Результат верифікація Server в Web-інтерфейсі

4.1.3 Верифікація Firewall

Переміщуючись у робочий каталог ~/diploma і запускаючи команду terraform apply, ми спостерігаємо в консолі процес створення, а та проводимо верифікацію Firewall (рисунок 4.7).

```
Plan: 4 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

google_compute_firewall.allow_healthcheck: Creating...
google_compute_firewall.allow_internal_access: Creating...
google_compute_firewall.ext_eccess_bastion: Creating...
google_compute_firewall.port_80: Creating...
google_compute_firewall.allow_healthcheck: Still creating... [10s elapsed]
google_compute_firewall.allow_internal_access: Still creating... [10s elapsed]
google_compute_firewall.ext_eccess_bastion: Still creating... [10s elapsed]
google_compute_firewall.port_80: Still creating... [10s elapsed]
google_compute_firewall.allow_healthcheck: Creation complete after 12s [id=projects/diploma-shevchenko/global/firewalls/allow-healthcheck]
google_compute_firewall.port_80: Creation complete after 12s [id=projects/diploma-shevchenko/global/firewalls/app-port-80]
google_compute_firewall.ext_eccess_bastion: Creation complete after 12s [id=projects/diploma-shevchenko/global/firewalls/ext-eccess-bastion]
google_compute_firewall.allow_internal_access: Creation complete after 13s [id=projects/diploma-shevchenko/global/firewalls/allow-internal-access]

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

diploma_bucket_state = "diploma-shevchenko-state"
diploma_subnet_a = "diploma-shevchenko-subnet-a"
diploma_subnet_b = "diploma-shevchenko-subnet-b"
diploma_vpc_name = "projects/diploma-shevchenko/global/networks/diploma-shevchenko"
url_diploma_bucket_state = "https://www.googleapis.com/storage/v1/b/diploma-shevchenko-state"
alex@lenovo13:~/diploma$
```

Рисунок 4.7 — Результат верифікація Firewall в консолі

VPC firewall rules

Firewall rules control incoming or outgoing traffic to an instance. By default, incoming traffic from outside your network is blocked. Learn more

Note: App Engine firewalls are managed in the App Engine Firewall rules section.

SMTP port 25 disallowed in this project. Learn more

REFRESH | CONFIGURE LOGS | DELETE

Filter: Enter property name or value

Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network	Logs
default-allow-icmp	Ingress	Apply to all	IP ranges: 0.0.0.0/0	icmp	Allow	65534	default	Off
default-allow-internal	Ingress	Apply to all	IP ranges: 10.128.0.0/9	tcp:0-65535 udp:0-65535 icmp	Allow	65534	default	Off
default-allow-rdp	Ingress	Apply to all	IP ranges: 0.0.0.0/0	tcp:3389	Allow	65534	default	Off
default-allow-ssh	Ingress	Apply to all	IP ranges: 0.0.0.0/0	tcp:22	Allow	65534	default	Off
allow-healthcheck	Ingress	Apply to all	IP ranges: 209.85.204.0/22, 209.85.152.0/22, 169.25	all	Allow	1000	diploma-shevchenko	Off
allow-internal-access	Ingress	Apply to all	IP ranges: 10.0.0.0/8	all	Allow	1000	diploma-shevchenko	Off
app-port-80	Ingress	port-80	IP ranges: 0.0.0.0/0	tcp:80	Allow	1000	diploma-shevchenko	Off
ext-eccess-bastion	Ingress	Apply to all	IP ranges: 0.0.0.0/0	tcp:22	Allow	1001	diploma-shevchenko	Off

Рисунок 4.8 — Результат верифікація Firewall в Web-інтерфейсі

4.1.4 Верифікація NAT

Переходимо у робочий каталог ~/diploma і запускаючи команду terraform apply, ми ініціюємо процес створення ресурсів, відображений у консолі, та одночасно проводимо верифікацію налаштувань NAT (рисунок 4.9).

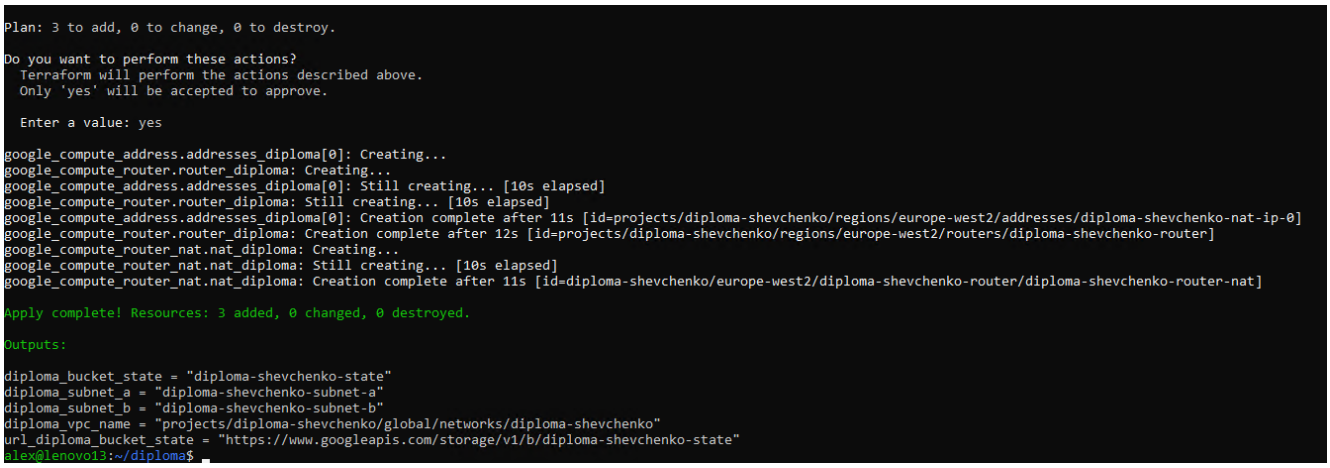

Рисунок 4.9 — Результат верифікація NAT в консолі

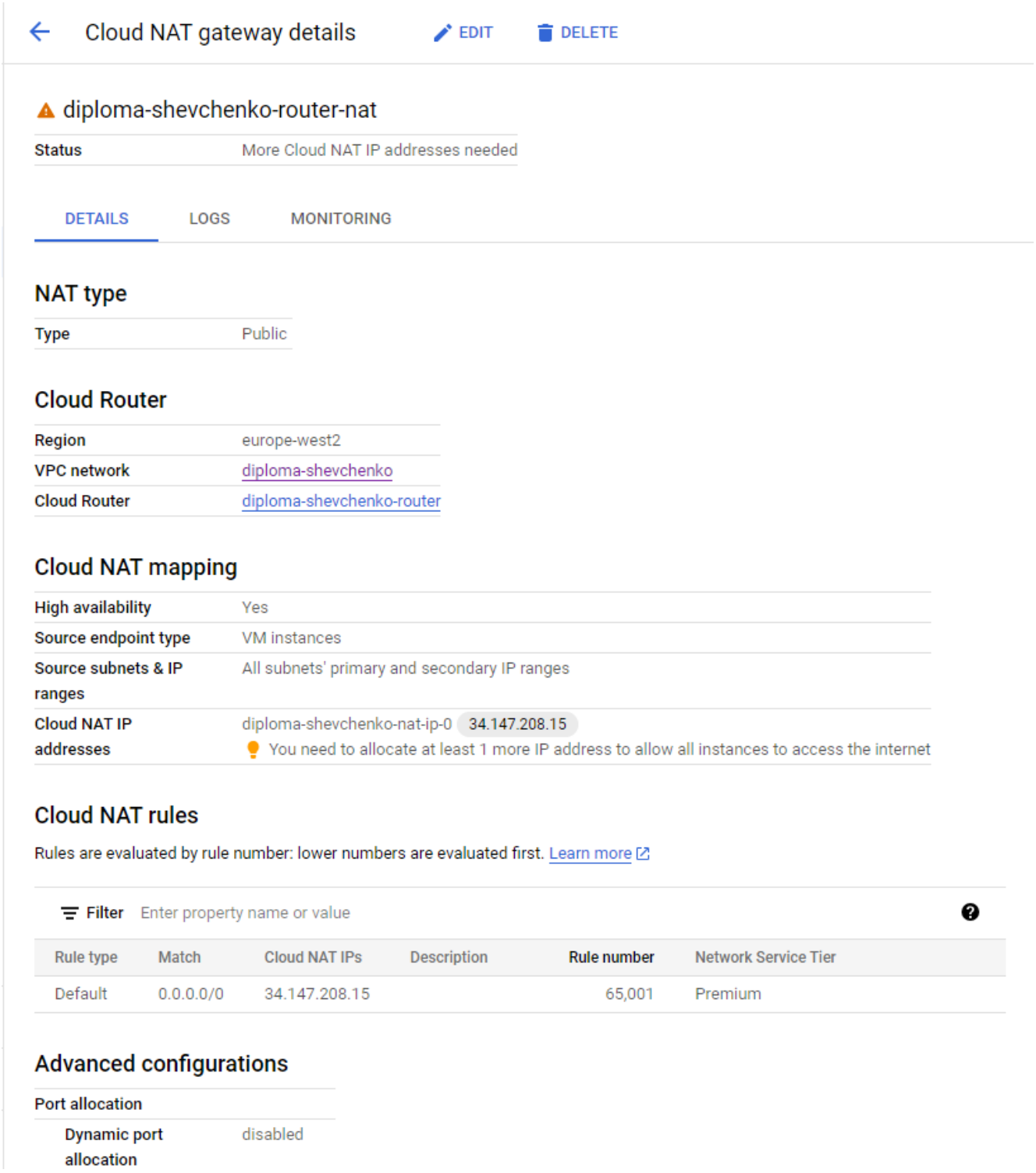

Рисунок 4.10 — Результат верифікація NAT в Web-інтерфейсі

4.1.5 Верифікація Load balancing

Переходячи у робочий каталог ~/diploma та виконуючи команду terraform apply, ми ініціюємо процес налаштування ресурсів, візуально відстежуючи цей процес у консолі, зокрема налаштування Load balancing (рисунок 4.11).

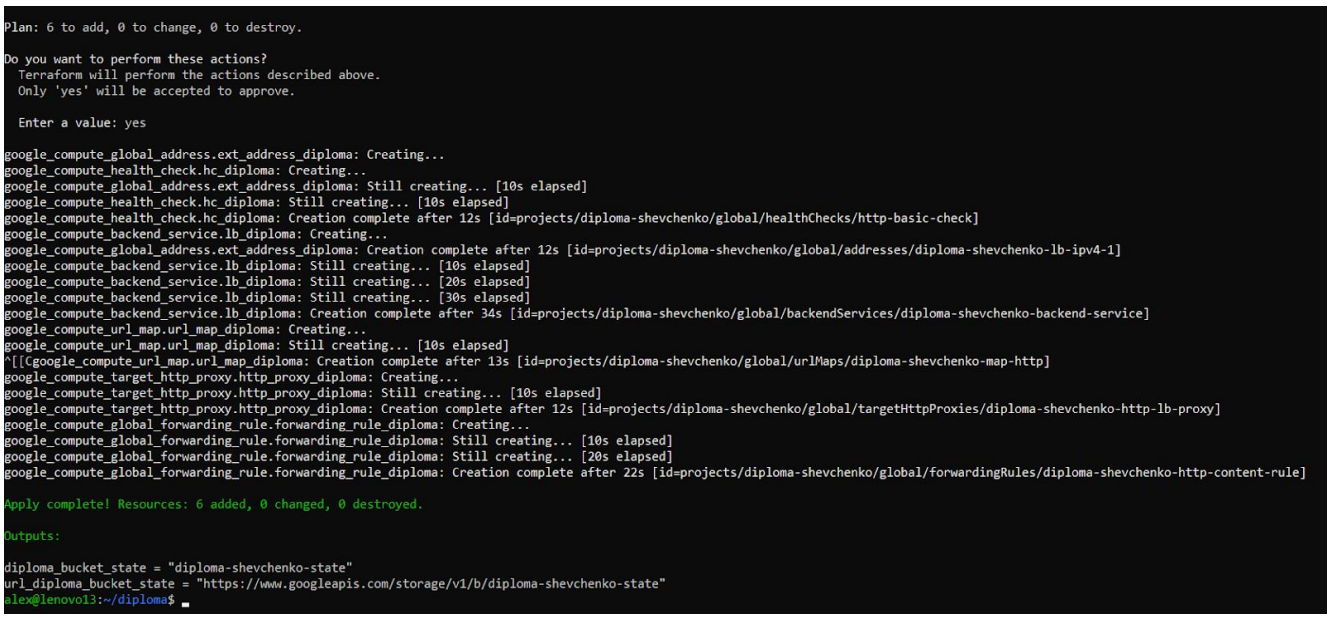

Рисунок 4.11 — Результат верифікація Load balancing в консолі

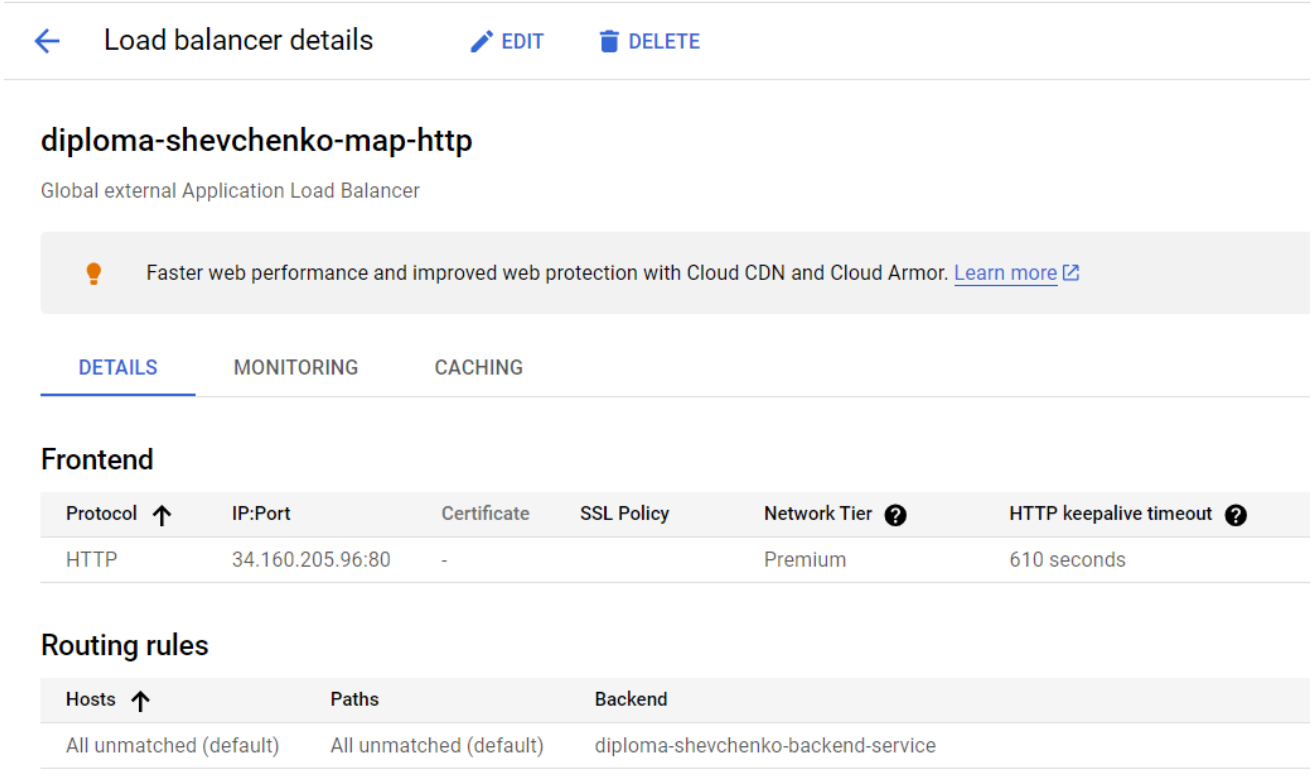

Рисунок 4.12 — Результат верифікація Load balancing в Web-інтерфейсі

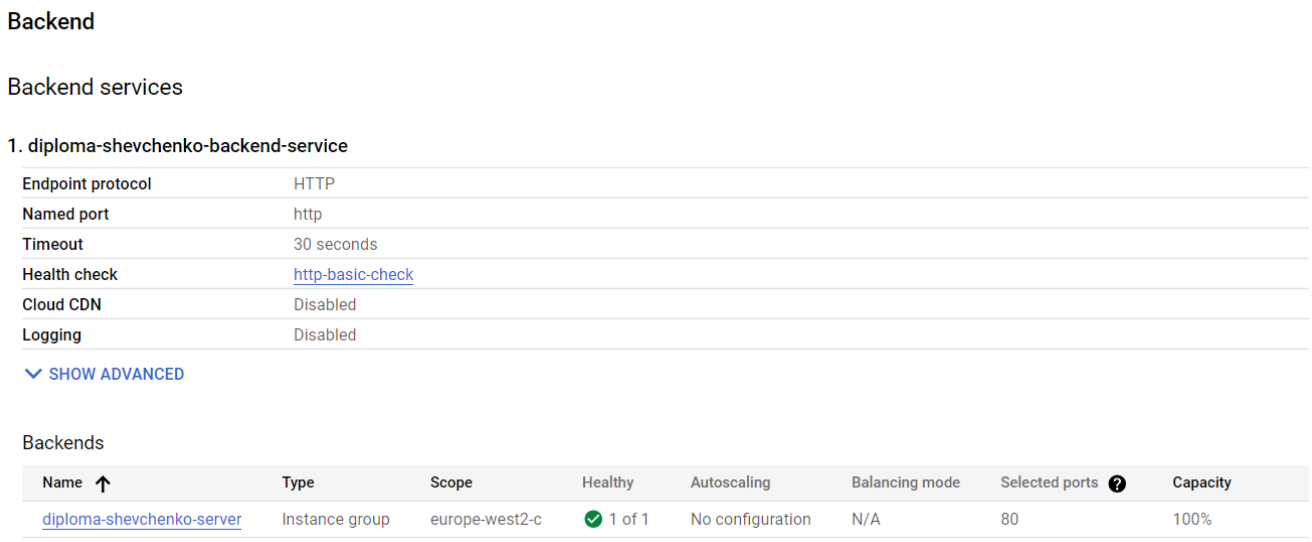

Рисунок 4.13 — Результат верифікація Load balancing backend в Web-інтерфейсі

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково–технічної діяльності.

Магістерська кваліфікаційна робота за темою “Технологія автоматизації розгортання обчислювальної інфраструктури в хмарному середовищі” передбачає розробку обчислювальної інфраструктури в залежності від потреб замовника.

Проведемо оцінювання комерційного потенціалу даної розробки. Для проведення технологічного аудиту було залучено 3–х незалежних експертів: Захарченко Сергій Михайлович – керівник магістерської кваліфікаційної роботи, професор, викладач кафедри комп’ютерних наук; Кадук Олександр Володимирович, Крупельницький Леонід Віталієвич.

Аудит науково-технічної розробки та її комерційного потенціалу проведено за допомогою таблиці 5.1, застосовуючи п’ятибальну шкалу оцінювання за 12-ма критеріями оцінки.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
#	0	1	2	3	4
1	2	3	4	5	6
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтвердже на розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогі в на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 5.1

1	2	3	4	5	6
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи:					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність:					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Закінчення таблиці 5.1

1	2	3	4	5	6
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки потрібно звести в таблицю за зразком таблиці 5.2.

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1 – Захарченко С.М.	2 – Кадук О.В.	3 – Крупельницький Л.В.
	Бали, виставлені експертами:		
1	3	4	2
2	2	3	3
3	3	3	2
4	2	2	4
5	4	3	3
6	3	2	3
7	4	4	3
8	3	2	2
9	4	4	3
10	3	3	3
11	4	3	3
12	3	2	2
Сума балів	$СБ_1=38$	$СБ_2=35$	$СБ_3=33$
середньоарифметична сума балів СБ	$СБ=13СБі3=38+35+333=34,3$		

В таблиці 5.3 наведено шкалу оцінки комерційного потенціалу розробки.

За результатами розрахунків, наведених в таблиці 5.2 та шкалою оцінки наведеної в таблиці 5.3 можна зробити висновок щодо рівня комерційного потенціалу розробки. Середньоарифметична сума балів, виставлених експертами склала 34.3, що відповідає рівню «вище середнього».

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Такий рівень комерційного потенціалу досягнуто за рахунок суттєвого зменшення витрат ресурсів і часу для автоматизованого розгортання та управління інфраструктурою, що значно знижуючи витрати на ручну працю та зменшуючи ризик людських помилок. Також, забезпечує краще використання ресурсів котрі надають клаудні провайдери, оскільки інфраструктура може бути швидко масштабована або модифікована залежно від потреб без необхідності фізичного втручання. Використання удосконалених алгоритмів розгортання інфратсруктури дозволяє підприємствам швидко адаптуватися до змін на ринку або у вимогах бізнесу, що в кінцевому підсумку призводить до економії коштів та збільшення доходів.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

Проведемо прогнозування витрат на виконання науково–дослідної, дослідно–конструкторської та конструкторсько–технологічної роботи для розробки програмного забезпечення, такі витрати, можна розрахувати за наступними статтями:

— розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи;

— розрахунок загальних витрат на виконання даної роботи;

— прогнозування загальних витрат на виконання та впровадження результатів даної роботи.

5.2.1 Основна заробітна плата

Для проектування та розгортання обчислювальної інфраструктури було залучено наступних спеціалістів:

— інфраструктурний інженер;

— керівник проєкту.

Посадові оклади, число днів роботи наведено в таблиці 5.4

Основна заробітна плата кожного із розробників (дослідників) $З_О$, якщо вони працюють в наукових установах бюджетної сфери розраховуються за формулою:

$$Зо=МТр*t \text{ грн}), \quad (5.1)$$

де M — місячний посадовий оклад конкретного розробника, грн;

T_p — число робочих днів в місяці; приблизно $Т_р = 22$ дні;

t — число робочих днів, конкретного розробника

Таблиця 5.4 — Основна заробітна плата

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник проєкту	24000	1 090,91	17	18 545,45
Інфраструктурний інженер	39000	1 772,73	37	65 590,91
Всього				84 136,36

Зо=84136,36 (грн).

5.2.2 Додаткова заробітна плата

Додаткова заробітна плата $З_д$ всіх робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12%) від суми основної заробітної плати робітників та розраховується за формулою:

$$Зд=(Зо+ Зр) *Ндоп100\%, \quad (5.2)$$

$$З_д = 0,10 * 84\ 136,36 = 8\ 413,64 \text{ (грн)}.$$

5.2.3 Відрахування на соціальні заходи

До статті «Відрахування на соціальні заходи» належать відрахування внеску на загальнообов'язкове державне соціальне страхування, та для здійснення заходів щодо соціального захисту населення (ЄСВ — єдиний соціальний внесок).

Нарахування на заробітну плату дослідників та робітників розраховується як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$Зн = (Зо +Зр + Здоп) * Нзп100\%, \quad (5.3)$$

де Зо — основна заробітна плата розробників, грн.;

Зд — додаткова заробітна плата всіх розробників та робітників, грн.;

Зр — витрати на основну заробітну плату робітників, грн.;

НЗП — ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

$$Зн=84\ 136,36+8\ 413,64 *0.22=20361 \text{ грн.}$$

5.2.4 Амортизація обладнання

До даної статті включаються амортизаційні відрахування по кожному виду обладнання, устаткування яке використовувалось розгортання обчислювальної інфраструктури в хмарному середовищі.

$$Аобл= ЦбТв*твик12, \quad (5.4)$$

де Ц — загальна балансова вартість всього обладнання, грн;

твик — час користування;

Тв — термін використання обладнання (приміщень), цілі місяці.

Амортизаційні відрахування наведено в таблиці 5.5.

Таблиця 5.5 — Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук HP	31500	5	2	1 050,00
Ноутбук Acer	27000	5	2	900,00
Офісне приміщення	250000	10	2	4 166,67
Всього				6 116,67

Аобл=6 116,67 (грн).

5.2.5 Витрати на силову електроенергію

До даної статті відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$Ве = В * П * Ф * Кп , \quad (5.5)$$

де В — вартість за 1 кВт, грн.;

П — установлена потужність обладнання, кВт;

Ф — фактична кількість годин роботи обладнання, годин;

Кп — коефіцієнт використання потужності;

Вартість електроенергії становить 7.2 грн за кВт в 2023 році.

Кп = Повна потужність (ВА) / Активна потужність (Вт).

Для кожного обладнання вона буде різна, отже порахуємо все і візьмемо середнє.

Кп.ноутбук1 =235 / 300 = 0,78;

Кп.ноутбук2 =215 / 300 = 0,71;

Кп.світло = 51 / 60 =0,85 ;

Кп.конд = 710 / 800 = 0,88;

Кп = (0,78 + 0,71 + 0,85 + 0,88) / 4 = 0,8;

Розрахуємо скільки часу працювало все обладнання по окремості згідно термінів використання наведених у таблиці 5.6.

Таблиця 5.6 — Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук HP	0,3	352	760,32
Ноутбук Acer	0,3	352	760,32
лампи світодіодні	0,06	352	152,064
кондиціонер	0,8	240	1382,4
Всього	1,46	1296	3055,104

Сумарні витрати на електроенергію становлять:

Ве = 7,2 * 1,4 * 1296 * 0,8 = 10 450,94 (грн).

5.2.6 Загальні витрати

Сума всіх попередніх статей витрат дає витрати на виконання даної частини розділу роботи Взаг.

$$Взаг= Зо+Зд + Зн + Аобл+ Ве , \quad (5.6)$$

Взаг = 84136,36 + 8413,64 + 20361 + 6116,67 +10 450,94 = 129478,61 (грн)

де Зо — основна заробітна плата розробника, грн.;

Зд — додаткова заробітна плата розробника, грн.;

Зн — ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Аобл — амортизаційні відрахування, грн.;

Ве — сумарні витрати на електроенергію.

Розрахунок загальних витрат на виконання даної роботи.

$$Взаг=Вα[грн], \quad (5.7)$$

Взаг = 129478,61 / 2 = 64739,3 (грн).

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної роботи здійснюється за формулою:

$$ЗВ=Взагβ, \quad (5.8)$$

де β — коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Так, якщо розробка знаходиться:

— на стадії науково-дослідних робіт, то β ≈ 0,1;

— на стадії технічного проектування, то β ≈ 0,2;

— на стадії розробки конструкторської документації, то β ≈ 0,3;

— на стадії розробки технологій, то β ≈ 0,4;

— на стадії розробки дослідного зразка, то β ≈ 0,5;

— на стадії розробки промислового зразка, то β ≈ 0,7;

— на стадії впровадження, то β ≈ 0,9.

Коефіцієнт η є приблизним.

Отже, підставимо дані в формулу 5.8 й отримаємо результат:

ЗВ= 129478,61 / 0,7 = 92484,72 (грн).

Витрати на виконання наукової роботи та впровадження її результатів становитиме 92484,72 грн.

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У даному підрозділі проведемо кількісне прогнозування, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. В умовах ринку узагальнюючим позитивним результатом, що його отримує підприємство від впровадження результатів тієї чи іншої розробки, збільшення чистого прибутку підприємства. Зростання чистого прибутку можна оцінити у теперішній вартості грошей.

Зростання чистого прибутку забезпечить підприємству надходження додаткових коштів, які дозволять покращити фінансові результати діяльності.

Виконання даної наукової роботи та впровадження її результатів складає приблизно 1 рік.

Позитивні результати від впровадження розробки очікуються на другий рік впровадження.

Проведемо детальніше прогнозування позитивних результатів та кількісне їх оцінювання по роках. Обчислимо збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum^{n}_{1} (\Delta П_я * N + П_я * \Delta N)_n \text{ [грн]}, \quad (5.9)$$

де $\Delta П_я$ — покращення основного якісного показника від впровадження результатів розробки у даному році;

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ — основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

Припустимо, що внаслідок впровадження результатів наукової розробки покращується якість, що дозволяє підвищити ціну його реалізації на 150 грн, а кількість одиниць реалізованої послуги збільшиться:

— протягом першого року — на 0 од.;

— протягом другого року — на 4000 од.;

— протягом третього року — ще на 1500 од.

Орієнтовно: реалізація послуг до впровадження результатів наукової розробки складала 20 шт., а її ціна — 200 грн.

Спрогнозуємо збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства ΔП1 протягом першого року складе:

$$\Delta П_1 = 0 \text{ (грн)}.$$

Обчислимо збільшення чистого прибутку підприємства ΔП2 протягом другого року:

$$\Delta П_2 = (20 + 200) * 4000 = 880000 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства ΔП3 протягом третього року становитиме:

$$\Delta П_3 = (20 + 200) * (4000 + 1500) = 1210000 \text{ (грн)}.$$

Отже, розрахунки показують, що відповідно прогнозуванню комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку підприємства.

5.4 Визначення економічної доцільності фінансування розробки

Основними показниками, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахунок ефективності вкладених інвестицій передбачає:

1-й крок. Розрахунок теперішньої вартості інвестицій PV, що вкладаються в наукову розробку. Такою вартістю ми можемо вважати прогнозовану величину загальних витрат ЗВ на виконання та впровадження результатів НДДКР, тобто ЗВ = PV = 92484,72 (грн).

2-й крок. Розрахунок очікуваного збільшення прибутку $\Delta П_i$, що його отримає підприємство (організація) від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження проведено вище.

3-й крок. Приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$ПП = 1T\Delta Пi(1 + \tau)t, \qquad (5.10)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн.;

Т — період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні - 0,1;

t — період часу (в роках) від моменту отримання чистого прибутку до точки „0".

$$ПП = (92484,72/(1+0,1)^1) + (0/(1+0,1)^2)) + (880000/(1+0,1)^3) +$$
$$+ (1210000 /(1+0,1)^4)) = 92484,72/1,1 + 0 + 880000/1,331 +$$
$$+ 1210000/1,4641 = 84077,01 + 661157,02 + 826446,28 =$$
$$= 1571680,31 \text{ (грн)}.$$

Абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки розраховуємо за формулою:

$$Е_{абс} = ПП - PV \text{ [грн]}, \quad (5.10)$$

$$Е_{абс} = 1571680,31 - 92484,72 = 1479195,59 \text{ (грн)}.$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн.;

PV – теперішня вартість початкових інвестицій, грн.

Оскільки $Е_{абс} > 0$, результат від проведення наукових досліджень щодо розробки програмного продукту та їх впровадження принесе прибуток, тобто є доцільним, але це ще не свідчить про те, що інвестор буде зацікавлений у фінансуванні даної програми.

4-й крок. Внутрішня економічна дохідність інвестицій $Е_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науковотехнічної розробки. Розраховують відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$Ев = Тж1+ЕабсPV-1, \quad (5.11)$$

де $Е_{абс}$ — абсолютна ефективність вкладених інвестицій, грн.;

PV — теперішня вартість інвестицій PV = ЗВ, грн.;

$Т_ж$ — життєвий цикл наукової розробки, роки.

$$Ев = 41+1479195,5992484,72-1 = 41+15,99-1 =416,99 -1 = 2,03-1 = 1,03$$

або 103 %

Порівняємо $Е_в$ з мінімальною (бар'єрною) ставкою дисконтування мін, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть.

Спрогнозуємо величину $\tau_{мін}$. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$. визначається за формулою:

$$\tau = d + f, \quad (5.12)$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; d = 0,14;

f — показник, що характеризує ризикованість вкладень; величина f = 0,3.

$$\tau = 0,14 + 0,3 = 0,44$$

Припустимо, що за даних умов прибуток буде збільшуватись, то у інвестора є потенційна зацікавленість у фінансуванні даної наукової розробки.

5.4.1 Розрахунок терміну окупності вкладених у реалізацію наукового проекту інвестицій

Розраховують термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ за формулою:

$$T_{ок} = 1 / E_{в}[грн], \quad (5.13)$$

$$T_{ок} = 1 / 1,03 = 0,97 \text{ (роки)}$$

Оскільки термін окупності вкладених у реалізацію наукового проекту інвестицій менше трьох років ($T_{ок} < 3$ років), то фінансування нової розробки є доцільним.

5.5 Результати економічного аналізу

У цьому розділі було проведено оцінку комерційних перспектив розробки технологій автоматизації розгортання обчислювальної інфраструктури в хмарному середовищі.

Під час технологічного аудиту, у якому брали участь три експерти, виявлено, що комерційний потенціал цієї розробки перевищує середні показники. Розглядаючи комерційний потенціал, стало зрозуміло, що програмний продукт перевершує своїх конкурентів і має високі перспективи, зокрема завдяки кращим функціональним можливостям, що робить його конкурентоспроможним на ринку.

Загальні витрати на наукові та технологічні дослідження склали 92484,72 грн, в той час як оцінка абсолютної ефективності інвестицій на суму 1479195,59 грн вказує на потенційний прибуток для інвестора від впровадження цього продукту.

Щорічна ефективність інвестицій у наукову розробку становить 103%, що значно вище за мінімальну бар'єрну ставку дисконтування у 44%, що свідчить про високий інтерес інвесторів до фінансування проекту. Термін окупності інвестицій становить менше одного року (0,97 року), що також підтверджує доцільність фінансування.

Всі ці фактори разом створюють міцну основу для рішення про розробку і виробництво нового продукту.

ВИСНОВКИ

У цій роботі було зосереджено увагу на аналізі передових технологій Infrastructure as Code (IaC) для ефективного автоматизованого розгортання обчислювальних інфраструктур у хмарному середовищі. Вибрано хмарну платформу Google Cloud Platform в якості зручності користування для розгортання. Для реалізації інфраструктури обрано мову HashiCorp Configuration Language та інструмент HashiCorp Terraform, який відзначається популярністю, гнучкістю та ефективністю.

У дослідженні вдосконалено методологію створення та оновлення інформаційних ресурсів. Це дозволило значно оптимізувати процеси розгортання нових інфраструктур, зокрема через швидке додавання чи видалення обчислювальних ресурсів до інформаційної системи. Ця оптимізація забезпечила економію часу та коштів на ресурси.

Проект включав практичну реалізацію запропонованого методу. Розроблене рішення дозволяє не тільки створювати та оновлювати інфраструктуру, але й ефективно модифікувати та видаляти її компоненти. Верифікація працездатності розробленого рішення підтвердила його надійність та ефективність.

Додатково було проведено економічний аналіз розробленого рішення. Результати аналізу показали, що рішення не лише економічно вигідне, але й конкурентоспроможне на ринку, забезпечуючи значні переваги для організацій, які його використовують.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Технологія автоматизації розгортання інфраструктури в хмарному середовищі [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/schedConf/presentations?searchField=&searchMatch=&search=&track=857

2. Mastering KVM Virtualization - by Humble Devassy Chirammal, Prasad Mukhedkar, Anil Vettathu - August 19, 2016

3. Infrastructure as Code for Beginners, by Russ McKendrick - 222 pages, May 2023

4. Infrastructure as Code (IAC) Cookbook, by Stephane Jourdan, Pierre Pomes - 347 pages, February 17, 2017

5. Programming languages [Електронний ресурс]. Режим доступу: https://octoverse.github.com/2022/top-programming-languages

6. What is Infrastructure as Code [Електронний ресурс]. Режим доступу: https://aws.amazon.com/ru/what-is/iac/

7. Google Cloud overview [Електронний ресурс]. Режим доступу: https://cloud.google.com/docs/overview/

8. Programming languages [Електронний ресурс]. Режим доступу: https://octoverse.github.com/2022/top-programming-languages

9. Reports languages [Електронний ресурс]. Режим доступу: https://unicode.org/reports/tr31/

10. Terraform vs. Ansible: Key Differences and Comparison of Tools [Електронний ресурс]. Режим доступу: https://k21academy.com/ansible/terraform-vs-ansible/

11. Syntax HCL [Електронний ресурс]. Режим доступу: https://developer.hashicorp.com/terraform/language/syntax/configuration

12. What is it infrastructure [Електронний ресурс]. Режим доступу: https://www.freecodecamp.org/news/what-is-it-infrastructure/

13. Get Started - Google Cloud [Електронний ресурс]. Режим доступу: https://developer.hashicorp.com/terraform/tutorials

14. Terraform in Action, by Scott Winkler - 408 pages, July 6, 2021 by Manning

15. What is Terraform and HashiCorp Developer [Електронний ресурс]. Режим доступу: https://www.terraform.io/intro

16. Build infrastructure [Електронний ресурс]. Режим доступу: https://developer.hashicorp.com/terraform/tutorials/gcp-get-started/google-cloud-platform-build

17. Cloud SDK [Електронний ресурс]. Режим доступу: https://cloud.google.com/sdk?hl=en

18. Infrastructure as Code: Managing Servers in the Cloud, by Kief Morris - 336 pages, July 26, 2016 by O'Reilly Media

19. Resource management [Електронний ресурс]. Режим доступу: https://cloud.google.com/docs/terraform/resource-management/store-state

20. Terraform: Up and Running, 3rd Edition, by Yevgeniy Brikman- 618 pages, September 2022

21. Infrastructure as Code: Dynamic Systems for the Cloud Age 2nd Edition, by Kief Morris - 430 pages, O'Reilly, 2021

22. What is infrastructure as code [Електронний ресурс]. Режим доступу: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac

23. Tutorial IaC [Електронний ресурс]. Режим доступу: https://cloud.google.com/recommender/docs/tutorial-iac

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н.. Азаров О.Д..
"29" вересня 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

"Технологія автоматизації розгортання інфраструктури в хмарному середовищі"

08-54.МКР.047.00.000 ТЗ

Науковий керівник к.т.н., проф. каф. ОТ
_____ Захарченко С.М.
Студент групи 2КІ-22м
_____ Шевченко О.В.

ВНТУ 2023

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Важливим є актуальність дослідження у напрямку магістерської роботи є значимість хмарної інфраструктури в сучасному бізнес-середовищі. Вона спрямована на допомогу бізнесам у розумінні того, як інтегрувати хмарні рішення для підвищення стійкості та гнучкості їхньої IT-інфраструктури, що є ключовим аспектом для забезпечення конкурентоспроможності та адаптивності на ринку. Особливу увагу приділено зниженню витрат на IT-інфраструктуру через використання інфраструктури як коду, що дозволяє виявити нові стратегії для економічно-ефективного застосування. Можливості автоматизації можуть сприяти швидкій адаптації компаній до змінних вимог ринку. В цілому на меті надати компаніям вказівки щодо оптимізації використання хмарного сховища та додатків для підвищення ефективності бізнес-процесів.

1.2 Наказ про затвердження теми МКР.

2 Мета МКР і призначення розробки

2.1 Мета роботи — метою даної магістерської роботи є мінімізація витрат часу на розгортання обчислювальної інфраструктури в хмарному середовищі шляхом створення програмного коду для керування цим процесом.

2.2 Призначення розробки — визначається необхідністю створення ефективного методу на розгортання інфраструктури в хмарному середовищі шляхом створення програмного коду керування.

3 Вихідні дані для виконання МКР

3.1 Проведення аналізу наявних методів та принципів.

3.2 Розробка модулів для розгортання інфраструктури в хмарному середовищі.

3.4 Проведення верифікації та аналізу отриманих результатів.

3.5 Виконання розрахунків для доведення доцільності нової розробки з економічної точки зору.

4 Вимоги до виконання МКР

Головна вимога — що модулі забезпечують декларативний підхід для розгортання інфраструктури в хмарному середовищі.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи МКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Огляд архітектури Google Cloud Platform та HashiCorp Terraform модулів	18.09.2023	25.09.2023	Аналітичний огляд джерел, задачі досліджень, розділ 1
2	Технологія Infrastructure as Code (IaC)	28.09.2023	5.10.2023	Розділ 2
3	Розробка Infrastructure as Code та розгортання в Google Cloud Platform	5.10.2023	12.10.2023	Розділ 3
4	Підготовка економічної частини	12.10.2023	19.10.2023	Розділ 4
5	Апробація та впровадження результатів дослідження	18.10.2023	26.10.2023	Тези доповідей
6	Оформлення пояснювальної записки, графічного матеріалу і презентації	27.10.2023	2.11.2023	ПЗ, графічний матеріал і презентація
7	Підготовка і підпис супроводжуючих документів, нормоконтроль та тест на плагіат	2.11.2023	10.11.2023	Оформленні документи

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового

керівника, відгук опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлюванні МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104–2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документи на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ–03.02.02 П.001.01:21.

ДОДАТОК Б

Модель графа залежностей ресурсів

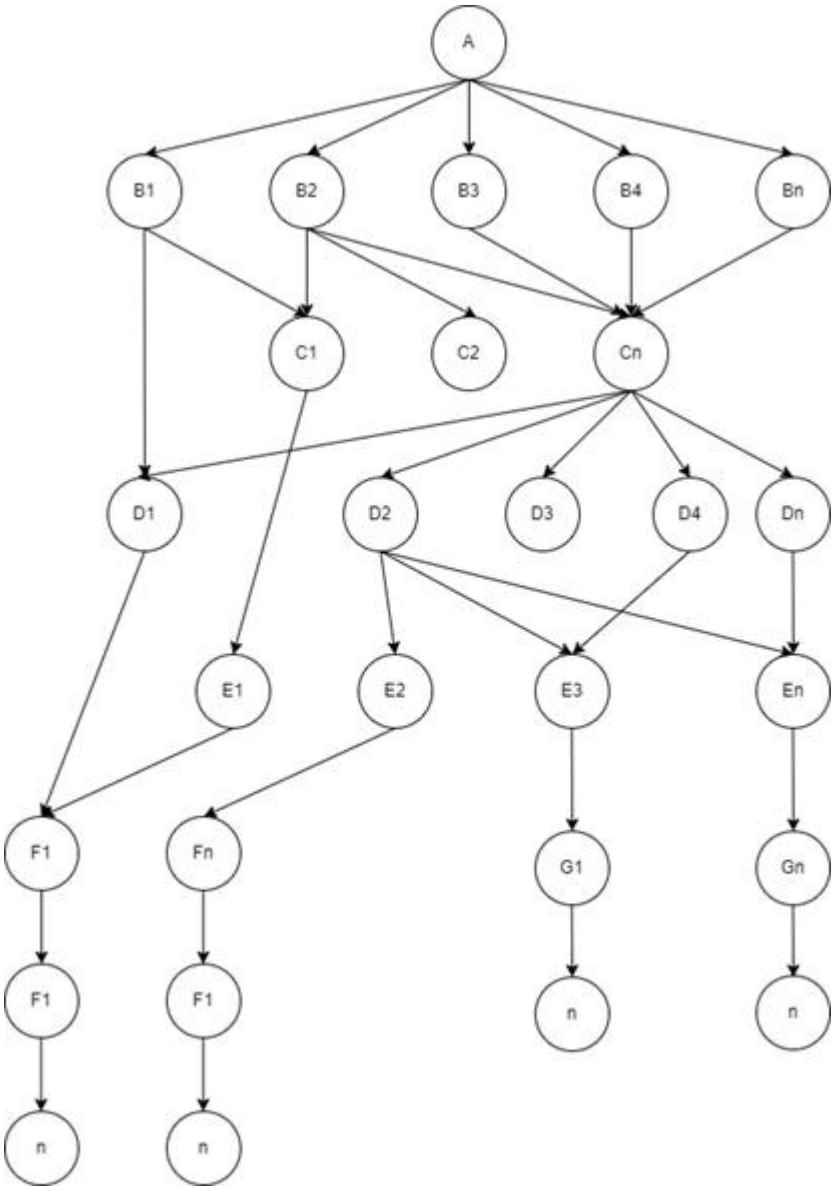

Рисунок Б.1 — Модель графа залежностей ресурсів

ДОДАТОК В

Схема залежностей інфраструктурних компонентів

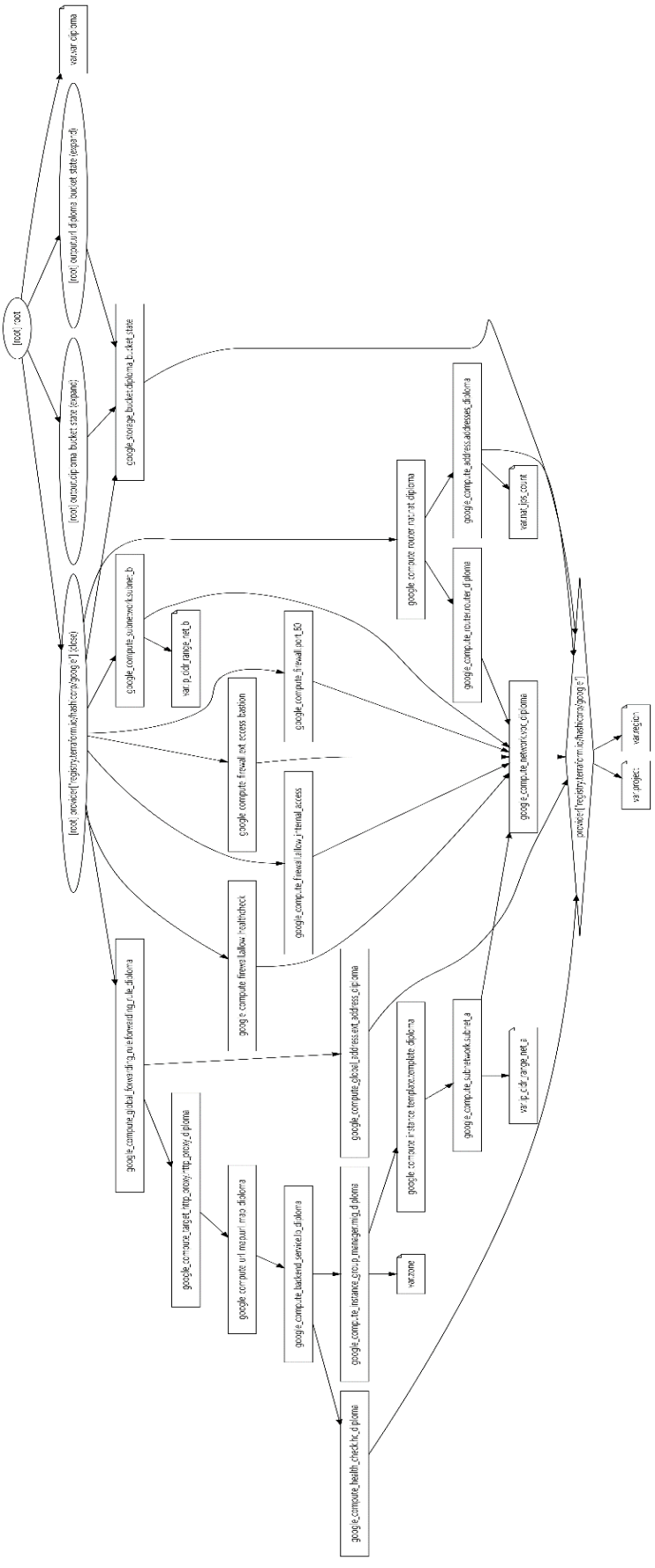

Рисунок В.1 — Схема залежностей інфраструктурних компонентів

ДОДАТОК Г

Життєвий цикл інфраструктури

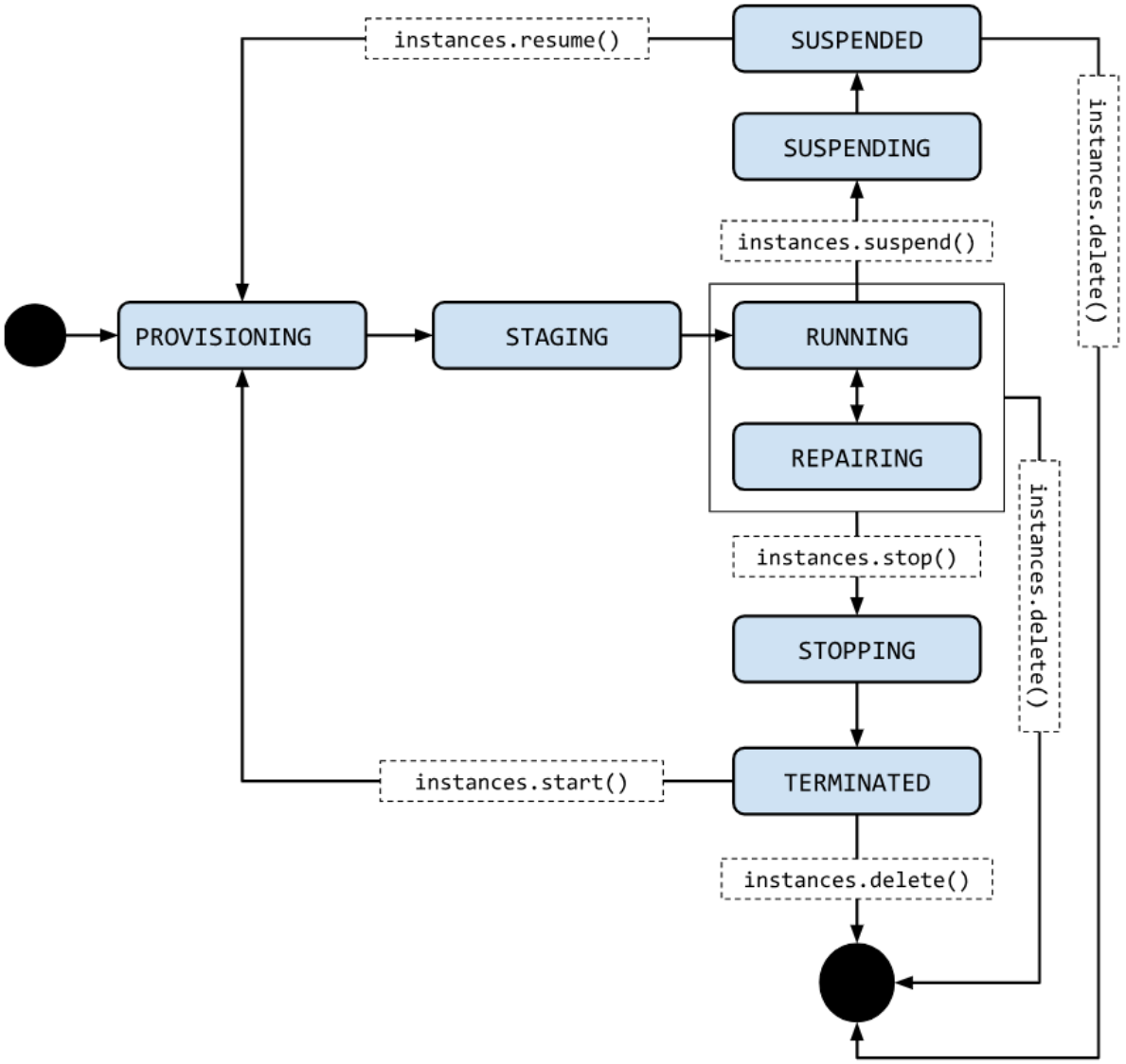

Рисунок Г.1 — Життєвий цикл інфраструктури

ДОДАТОК Д

Алгоритм управління інфраструктурою, гілка Create

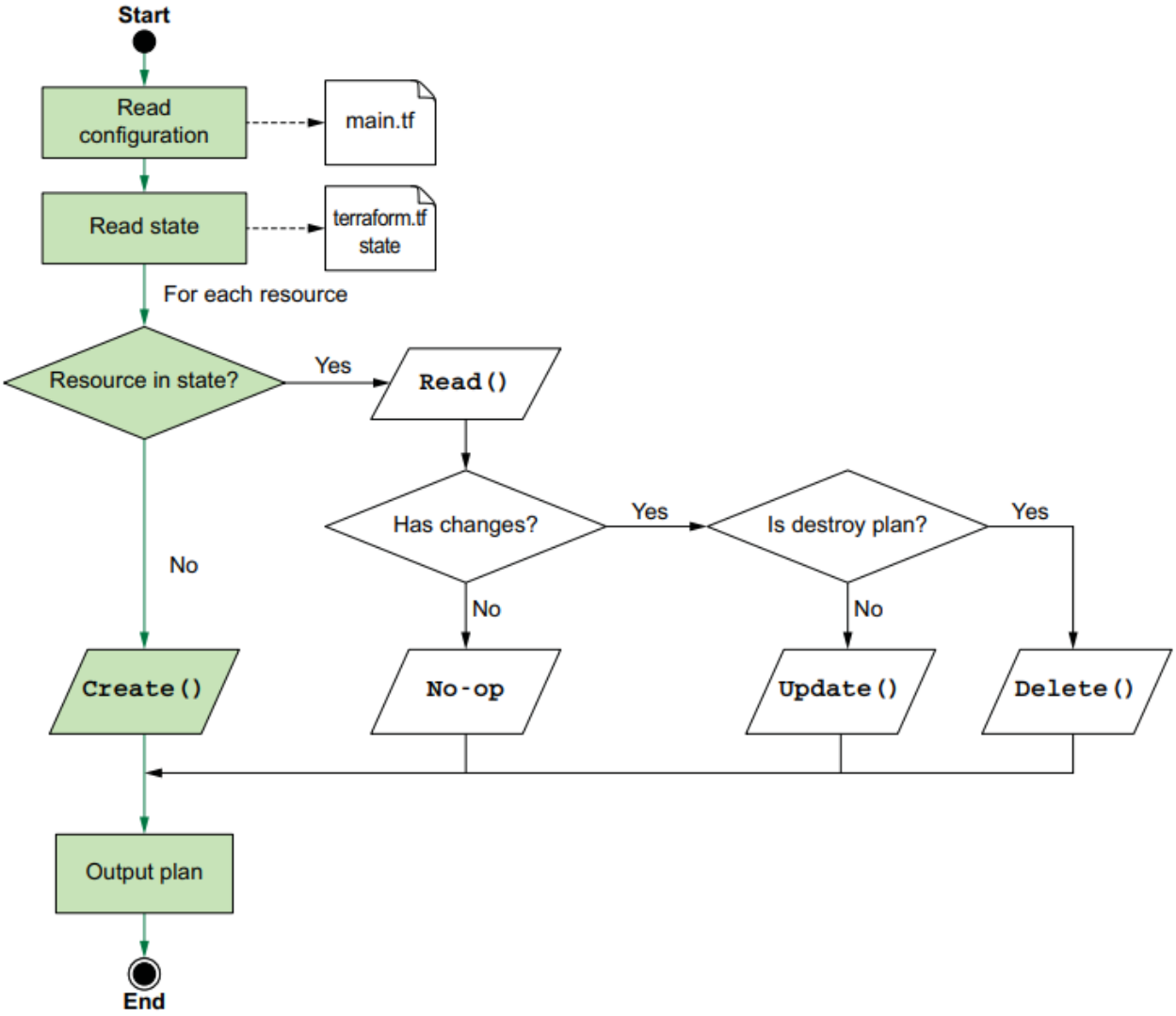

Рисунок Д.1 — Алгоритм управління інфраструктурою, гілка Create

ДОДАТОК Е

Алгоритм управління інфраструктурою, гілка Read

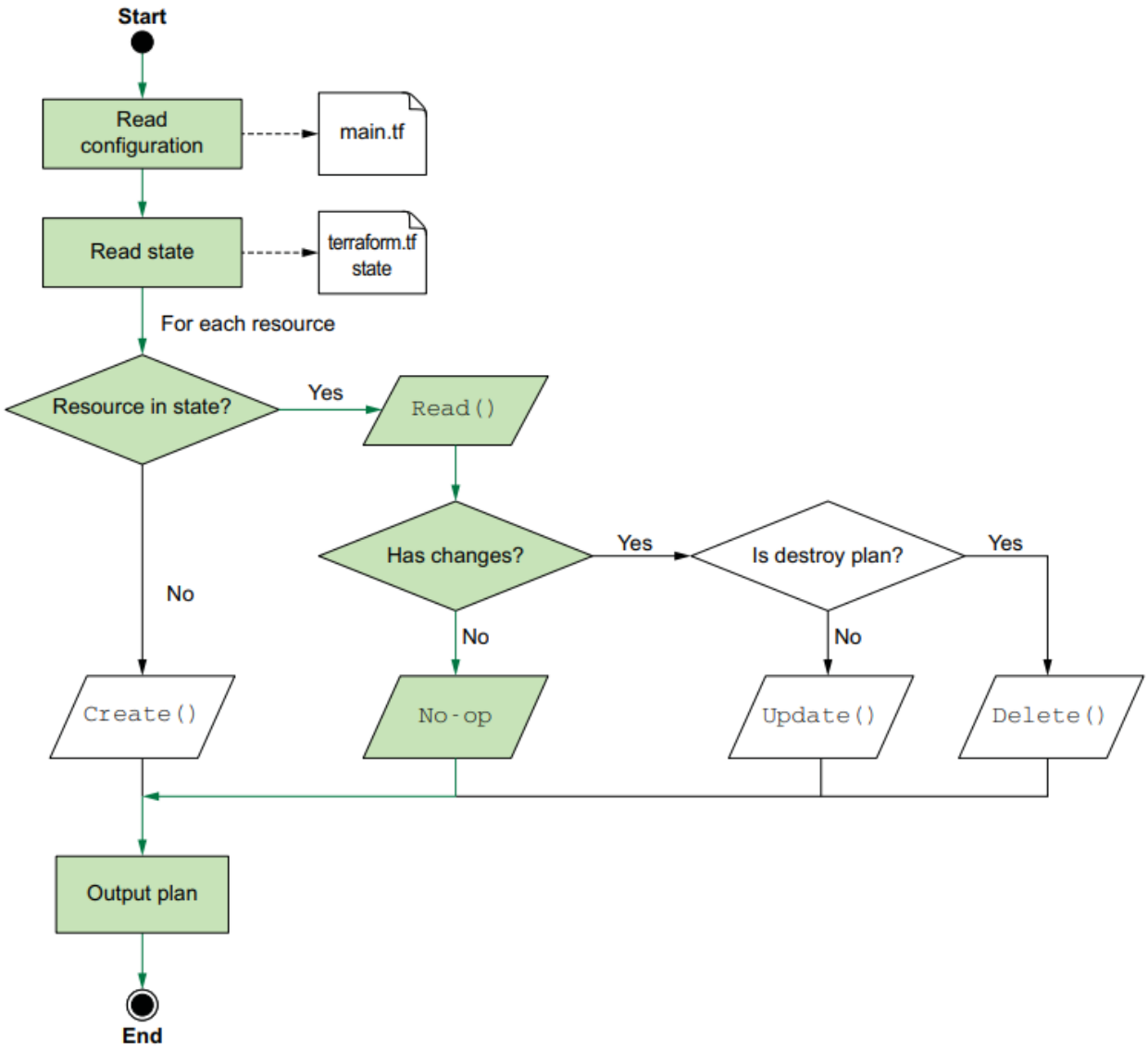

Рисунок Е.1 — Алгоритм управління інфраструктурою, гілка Read

ДОДАТОК Ж

Алгоритм управління інфраструктурою, гілка Update

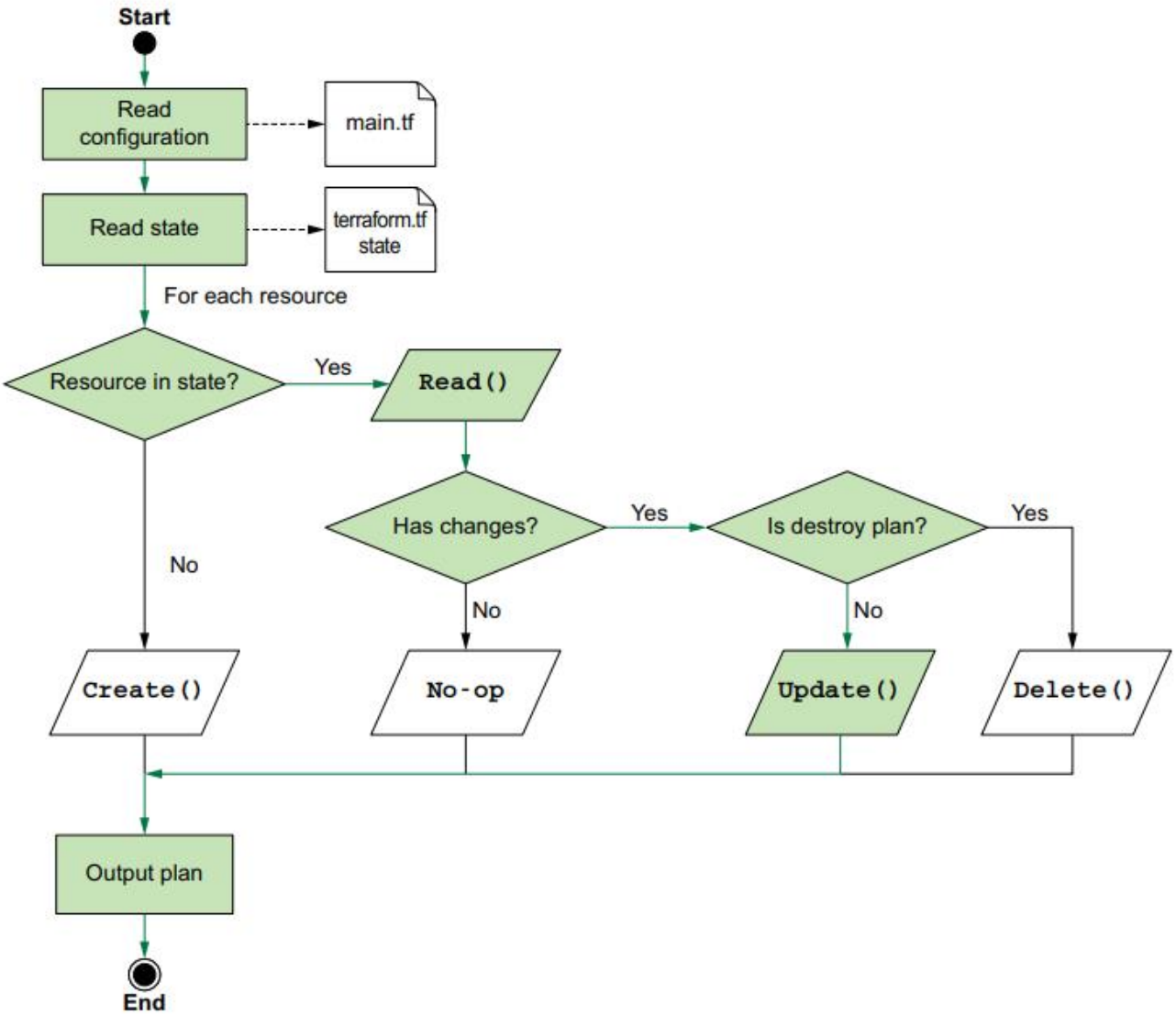

Рисунок Ж.1 — Алгоритм управління інфраструктурою, гілка Update

ДОДАТОК И

Алгоритм управління інфраструктурою, гілка Delete

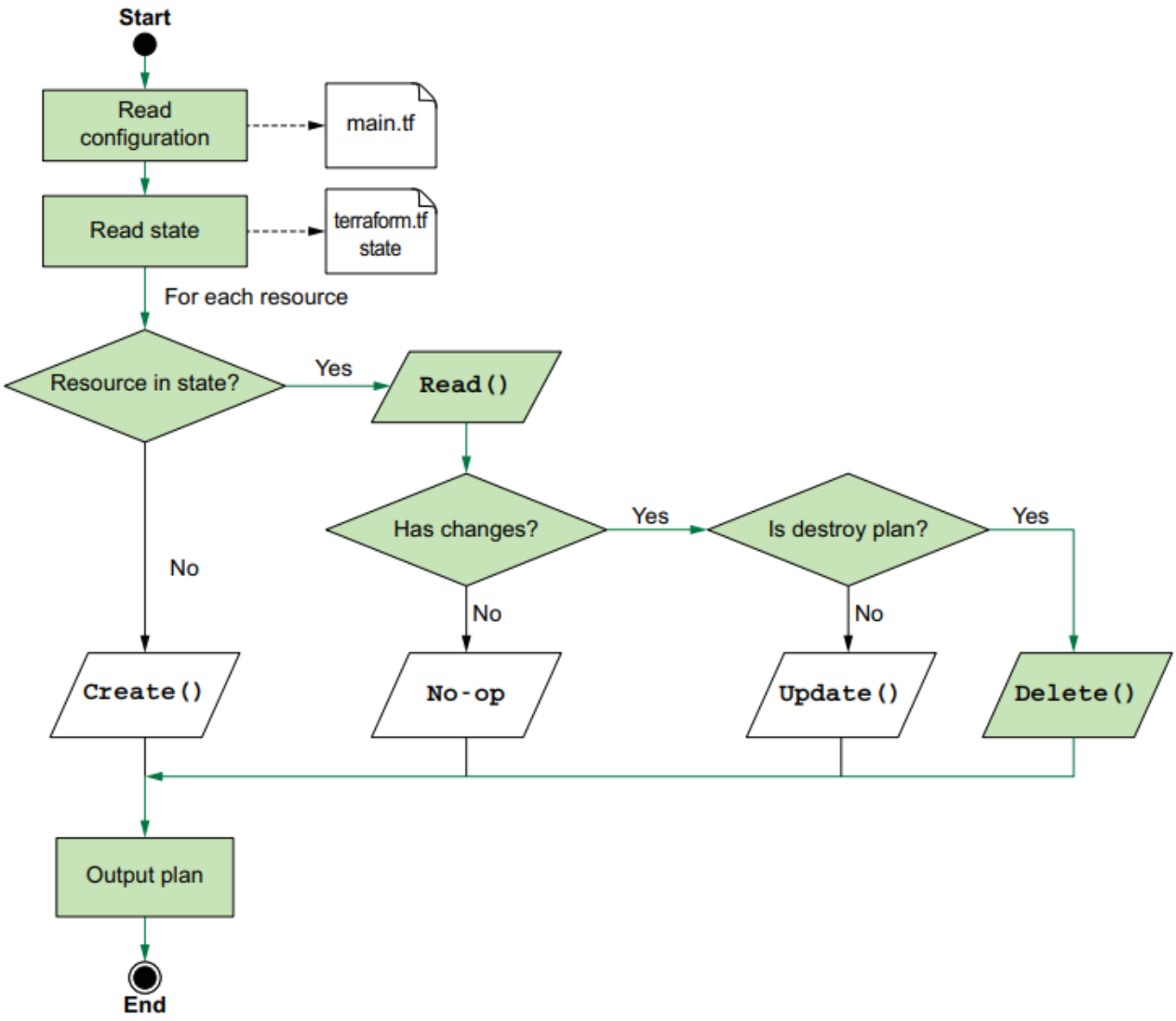

Рисунок И.1 — Алгоритм управління інфраструктурою, гілка Delete

ДОДАТОК К

Лістинг програми

_google.tf

```
# Specify the GCP Provider
provider "google" {
    project = var.project
    region  = var.region
}
```

_bucket.tf

```
# Create a GCS Bucket
resource "google_storage_bucket" "diploma_bucket_state" {
  name          = "diploma-shevchenko-state"
  force_destroy = false
  location      = var.region
  storage_class = "STANDARD"
  versioning {
    enabled = true
  }
}
output "url_diploma_bucket_state" {
  description = "bucket for state"
  value       = google_storage_bucket.diploma_bucket_state.self_link
}
```

_vpc.tf

```
# VPC
```

```
resource "google_compute_network" "vpc_diploma" {
  provider                = google
  project                 = var.project
  name                    = "${var.project}"
  auto_create_subnetworks = "false"
}
# Subnet subnet_a
resource "google_compute_subnetwork" "subnet_a" {
  provider      = google
  project       = var.project
  name          = "${var.project}-subnet-a"
  region        = var.region
  network       = google_compute_network.vpc_diploma.name
  ip_cidr_range = var.ip_cidr_range_net_a
}
# Subnet subnet_b
resource "google_compute_subnetwork" "subnet_b" {
  provider      = google
  project       = var.project
  name          = "${var.project}-subnet-b"
  region        = var.region
  network       = google_compute_network.vpc_diploma.name
  ip_cidr_range = var.ip_cidr_range_net_b
}
```

_servers_list.tf

```
# server VM
resource "google_compute_instance" "server" {
  count        = 10
```

```
  name         = "${var.project}-server-${count.index}"
  machine_type = "e2-standard-2"
  zone         = "${var.region}-${var.zone}"
  labels       = {
    component       = "${var.project}-server"
    name            = "server"
  }
  tags         = ["${var.project}-server"]
  boot_disk {
    initialize_params {
      image = "ubuntu-2004-focal-v20231101"
      size  = 50
      type  = "pd-ssd"
    }
  }
  network_interface {
    network    = google_compute_network.vpc_diploma.name
    subnetwork = google_compute_subnetwork.subnet_a.name
  }
}


_fw.tf
resource "google_compute_firewall" "allow_healthcheck" {
  provider = google
  name     = "allow-healthcheck"
  network  = google_compute_network.vpc_diploma.name
  allow {
    protocol = "all"
  }
```

```
  source_ranges = ["35.191.0.0/16", "130.211.0.0/22", "209.85.152.0/22",
"209.85.204.0/22", "169.254.169.254"]
}
resource "google_compute_firewall" "allow_internal_access" {
  provider = google
  name     = "allow-internal-access"
  network  = google_compute_network.vpc_diploma.name
  allow {
    protocol = "all"
  }
  source_ranges = ["10.0.0.0/8"]
}
resource "google_compute_firewall" "ext_eccess_bastion" {
  name       = "ext-eccess-bastion"
  network    = google_compute_network.vpc_diploma.name
  priority   = 1001
  allow {
    protocol  = "tcp"
    ports     = ["22"]
  }
    source_ranges = ["0.0.0.0/0"]
}
resource "google_compute_firewall" "port_80" {
  name          = "app-port-80"
  network       = google_compute_network.vpc_diploma.name
  allow {
    protocol    = "tcp"
    ports       = ["80"]
  }
  source_ranges = ["0.0.0.0/0"]
```

```
   target_tags   = ["port-80"]
}
```

_NAT.tf

```
# Router
resource "google_compute_router" "router_diploma" {
  provider = google
  name     = "${var.project}-router"
  region   = var.region
  network  = google_compute_network.vpc_diploma.id
}


# NAT Addresses
resource "google_compute_address" "addresses_diploma" {
  count  = var.nat_ips_count
  name   = "${var.project}-nat-ip-${count.index}"
  region = var.region
}


# NAT
resource "google_compute_router_nat" "nat_diploma" {
  provider                           = google
  name                               = "${var.project}-router-nat"
  router                             = google_compute_router.router_diploma.name
  region                             = var.region
  nat_ip_allocate_option             = "MANUAL_ONLY"
  nat_ips                            = google_compute_address.addresses_diploma.*.self_link
```

```
  source_subnetwork_ip_ranges_to_nat =
"ALL_SUBNETWORKS_ALL_IP_RANGES"
  min_ports_per_vm                   = 8000
  udp_idle_timeout_sec               = 100        # Timeout (in seconds) for UDP
connections. Defaults to 30s if not set.
  icmp_idle_timeout_sec              = 60         # Timeout (in seconds) for ICMP
connections. Defaults to 30s if not set.
  tcp_established_idle_timeout_sec   = 90         # Timeout (in seconds) for TCP
established connections. Defaults to 1200s if not set.
  tcp_transitory_idle_timeout_sec    = 90         # Timeout (in seconds) for TCP
transitory connections. Defaults to 30s if not set.
  tcp_time_wait_timeout_sec          = 90         # Timeout (in seconds) for TCP
connections that are in TIME_WAIT state. Defaults to 120s if not set.
  enable_endpoint_independent_mapping = false
  log_config {
    enable = true
    filter = "ERRORS_ONLY"
  }
}
```

_lb.tf

```
resource "google_compute_global_address" "ext_address_diploma" {
  name       = "${var.project}-lb-ipv4-1"
  ip_version = "IPV4"
}
resource "google_compute_health_check" "hc_diploma" {
  name               = "http-basic-check"
  check_interval_sec = 5
  healthy_threshold  = 2
```

```
  http_health_check {
    port               = 80
    port_specification = "USE_FIXED_PORT"
    proxy_header       = "NONE"
    request_path       = "/"
  }
  timeout_sec         = 5
  unhealthy_threshold = 2
}
resource "google_compute_backend_service" "lb_diploma" {
  name                            = "${var.project}-backend-service"
  connection_draining_timeout_sec = 0
  health_checks                   = [google_compute_health_check.hc_diploma.id]
  load_balancing_scheme           = "EXTERNAL_MANAGED"
  port_name                       = "http"
  protocol                        = "HTTP"
  session_affinity                = "NONE"
  timeout_sec                     = 30
  backend {
    group           =
google_compute_instance_group_manager.mig_diploma.instance_group
    balancing_mode  = "UTILIZATION"
    capacity_scaler = 1.0
  }
}
resource "google_compute_url_map" "url_map_diploma" {
  name            = "${var.project}-map-http"
  default_service = google_compute_backend_service.lb_diploma.id
}
resource "google_compute_target_http_proxy" "http_proxy_diploma" {
```

```
  name    = "${var.project}-http-lb-proxy"
  url_map = google_compute_url_map.url_map_diploma.id
}
resource "google_compute_global_forwarding_rule" "forwarding_rule_diploma" {
  name                  = "${var.project}-http-content-rule"
  ip_protocol           = "TCP"
  load_balancing_scheme = "EXTERNAL_MANAGED"
  port_range            = "80-80"
  target                = google_compute_target_http_proxy.http_proxy_diploma.id
  ip_address            = google_compute_global_address.ext_address_diploma.id
}
```

ДОДАТОК Л

Протокол перевірки кваліфікаційної роботи

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Технологія автоматизації розгортання інфраструктури в хмарному середовищі**.**
Тип роботи: кваліфікаційна робота
Підрозділ: кафедра обчислювальної техніки, ФІТКІ, 2–КІ–22м
Науковий керівник: к.т.н., проф. каф. ОТ. Захарченко С.М.

Unicheck	
Оригінальність	**94.1%**
Схожість	5.9%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи «Технологія автоматизації розгортання інфраструктури в хмарному середовищі».

Автор _____ Шевченко О.В.

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку Захарченко С.М.

_____ _____

(підпис) (прізвище, ініціали)

Експерт _____ _____

(за потреби) (підпис) (прізвище, ініціали, посада)