Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Розгортання та управління Kubernetes-кластерами за допомогою IaC

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-5
Марія ЦАПЕНКО
(власне ім'я, прізвище)

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

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

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Дар'я ТИМОШЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

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

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

Освітня програма Комп’ютерна інженерія
(повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Цапенко Марії Сергіївні
(прізвище, ім’я, по батькові)

1. Тема роботи Розгортання та управління Kubernetes-кластерами за допомогою IaC

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 2025 р.

3. Вхідні дані до роботи Тема дослідження
Офіційна документація Terraform, Ansible, Kubespray, Kubernetes, Helm та FluxCD
Helm-чарти системних компонентів

4. Перелік питань, що потрібно опрацювати у роботі
Аналіз теоретичних основ контейнеризації, оркестрації та підходу Infrastructure as Code
Інтеграція Terraform із Ansible для автоматизованого налаштування кластера Kubernetes
Встановлення системних компонентів кластеру через Helm (
Розробка шаблонів Terraform для автоматичного створення віртуальних машин, мереж,

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій _____

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.25-30.05.25	
2	Вибір технологій та інструментів	31.05.25-01.06.25	
3	Розробка конфігурацій	02.06.25-06.06.25	
4	Тестування та відлагодження	07.06.25-09.06.25	
5	Оформлення матеріалів кваліфікаційної роботи	10.06.25-11.06.25	
6	Подання кваліфікаційної роботи керівникові та	12.06.25-13.06.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на рецензу-	14.06.25-16.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Дар'я ТИМОШЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 52 с., 15 рис., 0 табл., 1 дод., 15 джерел.

TERRAFORM, ANSIBLE, KUBERNETES, КОНТЕЙНЕР, КЛАСТЕР, HELM, GITOPS, FLUXCD, ВІРТУАЛЬНА МАШИНА.

Метою кваліфікаційної роботи є розробка та практична реалізація інфраструктури комп'ютерної мережі з підтримкою безпечного доступу до Інтернету, маршрутизації трафіку, керування підключеними пристроями через Wi-Fi/WLAN, а також впровадження засобів фільтрації та контролю доступу з використанням протоколів та мережевих сервісів.

У ході виконання кваліфікаційної роботи було спроєктовано локальну комп'ютерну мережу з виходом до Інтернету через шлюз і маршрутизатор. У складі мережі налаштовано центральний сервер, який виконує функції DHCP, DNS та контролера доступу. Для забезпечення безпеки реалізовано міжмережевий екран (firewall), який обмежує вхідний та вихідний трафік згідно з визначеними правилами. Також впроваджено Wi-Fi-зону з використанням технології WLAN для підключення бездротових пристроїв. Особливу увагу приділено налаштуванню мережевих протоколів (TCP/IP, NAT, DHCP) та аналізу їх впливу на продуктивність мережі. В результаті було створено стабільну та масштабовану мережеву інфраструктуру, яка може бути використана як основа для подальшого розширення або інтеграції з корпоративними рішеннями

ABSTRACT

Bachelor’s thesis: 52 pages, 15 figures, 0 tables, 1 appendices, 15 sources.

TERRAFORM, ANSIBLE, KUBERNETES, CONTAINER, CLUSTER, HELM, GITOPS, FLUXCD, VIRTUAL MACHINE.

The major goal of this thesis is the development and practical implementation of a computer network infrastructure that supports secure Internet access, traffic routing, and device management via Wi-Fi/WLAN, along with the integration of filtering and access control mechanisms using modern protocols and network services.

In the course of the thesis, a local computer network was designed with Internet access provided through a gateway and a router. A central server was configured within the network to perform DHCP, DNS, and access control functions. To ensure network security, a firewall was implemented to restrict inbound and outbound traffic based on predefined rules. A Wi-Fi zone was also deployed using WLAN technology to enable wireless connectivity. Special attention was paid to the configuration of network protocols (TCP/IP, NAT, DHCP) and their impact on overall network performance. As a result, a stable and scalable network infrastructure was created, suitable for future expansion or integration with enterprise systems.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – програмний інтерфейс прикладного програмування (Application Programming Interface)

Ansible – система автоматизації конфігурації та розгортання

CIDR – класифікований міждоменний маршрут (Classless Inter-Domain Routing)

DNS – система доменних імен (Domain Name System)

FluxCD – система безперервного розгортання для Kubernetes

GitOps – підхід до автоматизації розгортання з використанням Git

Helm – менеджер пакетів для Kubernetes

IaC – інфраструктура як код (Infrastructure as Code)

K8s – Kubernetes – система оркестрації контейнерів

PVC – постійний том користувача (Persistent Volume Claim)

RBAC – керування доступом на основі ролей (Role-Based Access Control)

SSH – безпечний протокол доступу до систем (Secure Shell)

TF – Terraform – інструмент IaC для управління інфраструктурою

VM – віртуальна машина (Virtual Machine)

VPC – віртуальна приватна хмара (Virtual Private Cloud)

ВСТУП

Сучасна індустрія інформаційних технологій характеризується стрімким зростанням складності програмних систем та інфраструктури, необхідної для їх підтримки. Традиційні підходи до управління інфраструктурою, базовані на ручних операціях та скриптах, стають неефективними у контексті сучасних вимог до швидкості розробки, надійності та масштабованості систем. Контейнеризація додатків стала домінуючим трендом у розробці програмного забезпечення, а Kubernetes утвердився як стандарт де-факто для оркестрації контейнерів у продакшн-середовищах.

Проблематика ручного управління Kubernetes-кластерами стає особливо гострою при роботі з множинними середовищами, регіонами та командами розробників. Типові проблеми включають неконсистентність конфігурацій між середовищами, складність відтворення інфраструктури, відсутність версіонування змін, високий ризик людських помилок та значні часові витрати на підтримку. Ці виклики призводять до зниження продуктивності команд, збільшення часу виходу продукту на ринок та підвищення операційних ризиків.

Infrastructure as Code представляє революційний підхід до вирішення цих проблем, дозволяючи описувати, версіонувати та автоматизувати управління інфраструктурою через програмний код. Цей підхід забезпечує застосування принципів розробки програмного забезпечення до управління інфраструктурою, включаючи версіонування, тестування, рецензування коду та автоматизовану збірку. Особливо важливим є те, що IaC дозволяє досягти іммутабельності інфраструктури, коли замість модифікації існуючих ресурсів створюються нові версії з необхідними змінами.

Актуальність дослідження зумовлена кількома ключовими факторами. По-перше, зростаюча складність сучасних розподілених систем вимагає

більш досконалих підходів до управління інфраструктурою. По-друге, цифрова трансформація бізнесу створює потребу в більш швидкому та надійному розгортанні додатків. По-третє, культура DevOps та практики безперервної інтеграції та розгортання стають стандартом у індустрії, що вимагає автоматизації всіх аспектів життєвого циклу додатків, включаючи інфраструктуру.

Наукова новизна роботи полягає в комплексному дослідженні інтеграції принципів Infrastructure as Code з екосистемою Kubernetes, розробці методологічних підходів до проектування та впровадження IaC-рішень для управління контейнерною інфраструктурою, а також в аналізі практичних аспектів автоматизації повного життєвого циклу Kubernetes-кластерів від початкового розгортання до підтримки у продакшні.

Мета роботи полягає в дослідженні теоретичних основ та практичних методів застосування Infrastructure as Code для ефективного управління Kubernetes-кластерами, розробці комплексного підходу до автоматизації розгортання та підтримки контейнерної інфраструктури, а також в оцінці ефективності запропонованих рішень у реальних умовах експлуатації.

Для досягнення поставленої мети необхідно вирішити наступні завдання: провести аналіз сучасного стану розвитку технологій Infrastructure as Code та їх застосування для управління Kubernetes-інфраструктурою; дослідити архітектурні особливості Kubernetes та можливості його інтеграції з IaC-інструментами; розробити методологічний підхід до проектування IaC-рішень для Kubernetes-кластерів; створити практичне рішення для автоматизованого розгортання та управління Kubernetes-інфраструктурою; провести аналіз ефективності запропонованого підходу та надати рекомендації щодо його впровадження.

Об'єктом дослідження є процеси управління інфраструктурою Kubernetes-кластерів з використанням принципів Infrastructure as Code. Предметом дослідження є методи, інструменти та технології, що забезпечують автоматизоване розгортання, конфігурування та підтримку

Kubernetes-інфраструктури.

Методологічною основою дослідження є системний підхід до аналізу проблем управління інфраструктурою, методи проектування розподілених систем, принципи DevOps-культури та практики безперервної інтеграції та розгортання. Теоретичною базою роботи є наукові праці у галузі хмарних обчислень, контейнеризації додатків, автоматизації інфраструктури та управління конфігураціями.

Практична значущість роботи полягає в розробці конкретних рішень та рекомендацій для організацій, які впроваджують або планують впровадження Kubernetes у своїх технологічних стеках. Результати дослідження можуть бути використані для оптимізації процесів розробки та експлуатації контейнерних додатків, зниження операційних витрат та підвищення надійності систем.

1 ТЕОРЕТИЧНІ ОСНОВИ INFRASTRUCTURE AS CODE ТА KUBERNETES

1.1 Еволюція підходів до управління інфраструктурою

Історично управління IT-інфраструктурою пройшло кілька етапів еволюції, кожен з яких характеризувався специфічними підходами та інструментами. Ранній етап характеризувався повністю ручним управлінням, коли системні адміністратори фізично налаштовували сервери, встановлювали програмне забезпечення та вручну вносили зміни до конфігурацій. Цей підхід був прийнятним для невеликих інфраструктур, але швидко показав свої обмеження при зростанні масштабів систем.

Поява скриптової автоматизації ознаменувала перший крок до систематизації процесів управління інфраструктурою. Shell-скрипти, Perl та Python програми дозволили автоматизувати рутинні операції, але створили нові проблеми у вигляді складності підтримки та відсутності стандартизації. Кожна команда розробляла власні скрипти, що призводило до фрагментації знань та складнощів у передачі досвіду між проектами.

Розвиток інструментів управління конфігураціями, таких як Puppet, Chef та Ansible, представив концепцію декларативного опису бажаного стану системи. Ці інструменти дозволили абстрагуватися від специфічних кроків виконання та сконцентруватися на кінцевому результаті. Однак вони були орієнтовані переважно на управління існуючими серверами, а не на створення інфраструктури з нуля.

Хмарні обчислення кардинально змінили підходи до управління інфраструктурою, запровадивши концепцію програмного доступу до ресурсів через API. Це створило можливість трактування інфраструктури як коду, що можна створювати, модифікувати та видаляти програмно. AWS CloudFormation став одним з перших інструментів, які дозволили декларативно описувати хмарну інфраструктуру.

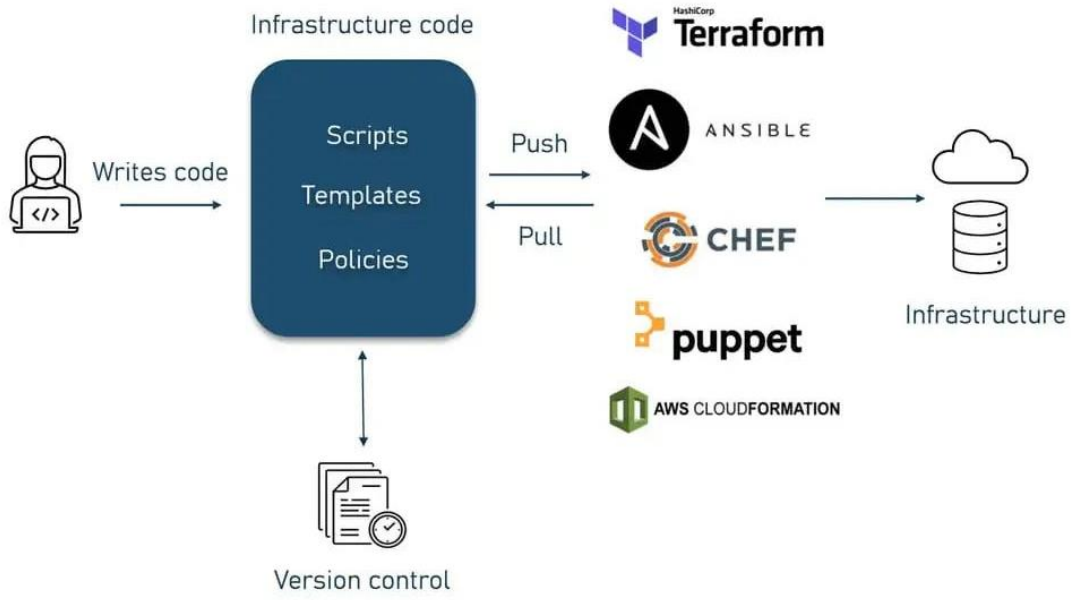

Рисунок 1.1 – Архітектура підходу Infrastructure as Code

Infrastructure as Code (рисунок 1.1) представляє логічне продовження цієї еволюції, об'єднуючи переваги декларативного підходу з можливостями хмарних платформ та найкращими практиками розробки програмного забезпечення. Цей підхід дозволяє застосовувати до інфраструктури ті самі принципи, що використовуються при розробці додатків: версіонування, тестування, рецензування коду, безперервну інтеграцію та розгортання.

1.2 Фундаментальні принципи Infrastructure as Code

Декларативність є однією з основоположних характеристик Infrastructure as Code, яка відрізняє його від традиційних імперативних підходів. Декларативний підхід передбачає опис бажаного стану інфраструктури без деталізації конкретних кроків для досягнення цього стану. Система автоматично визначає необхідні операції, порівнюючи поточний стан з описаним та виконуючи мінімально необхідний набір змін.

Перевагами декларативного підходу є простота розуміння та підтримки

конфігурацій, можливість повторного застосування тих самих описів для досягнення консистентного результату, а також природна ідемпотентність операцій. Ідемпотентність означає, що повторне застосування тієї самої конфігурації не призводить до небажаних змін або помилок.

Іммутабельність інфраструктури представляє парадигму, при якій інфраструктурні компоненти не змінюються після їх створення. Замість модифікації існуючих ресурсів створюються нові версії з необхідними змінами, після чого старі версії поступово виводяться з експлуатації. Цей підхід запозичений з функціонального програмування та надає значні переваги у терминах передбачуваності поведінки системи.

Іммутабельна інфраструктура спрощує процеси відновлення після збоїв, оскільки завжди можна швидко розгорнути відому робочу конфігурацію. Вона також знижує ризик накопичення "технічного боргу" в конфігураціях та забезпечує консистентність між різними середовищами. Drift detection стає простішим, оскільки будь-які відхилення від еталонного стану можуть бути легко виявлені та виправлені.

Версіонування інфраструктурного коду дозволяє застосовувати всі переваги систем контролю версій до управління інфраструктурою. Це включає можливість відстеження історії змін, розуміння того, хто, коли та чому вніс конкретні зміни, а також можливість швидкого повернення до попередніх версій у випадку проблем.

Branching strategies для інфраструктурного коду можуть відрізнятися від тих, що використовуються для додатків, але основні принципи залишаються схожими. Feature branches дозволяють експериментувати з новими конфігураціями без впливу на стабільні середовища, а pull requests забезпечують процес рецензування змін перед їх застосуванням.

Тестування інфраструктурного коду є критично важливим аспектом забезпечення якості. Unit testing може включати валідацію синтаксису конфігураційних файлів, перевірку відповідності політикам безпеки та тестування логіки умовних виразів. Integration testing перевіряє взаємодію

між різними компонентами інфраструктури, а acceptance testing підтверджує, що розгорнута інфраструктура відповідає функціональним вимогам.

1.3 Порівняльний аналіз інструментів Infrastructure as Code

Terraform займає провідні позиції серед інструментів IaC завдяки своїй універсальності та зрілості екосистеми. Його провайдерна архітектура дозволяє управляти ресурсами практично будь-якої хмарної платформи або сервісу через єдиний інтерфейс. Мова HCL забезпечує гарний баланс між читабельністю та експресивністю.

Сильними сторонами Terraform є детальне планування змін, що дозволяє попередньо оцінити наслідки застосування конфігурації. Модульна архітектура сприяє повторному використанню коду та стандартизації підходів. Великий репозиторій готових модулів значно прискорює розробку.

Слабкими сторонами Terraform є складність управління станом, особливо в командних середовищах. Обмеженість мови HCL може вимагати складних обхідних шляхів для реалізації деяких логічних конструкцій. Відсутність вбудованих механізмів тестування вимагає використання додаткових інструментів.

Pulumi пропонує альтернативний підхід, дозволяючи використовувати повноцінні мови програмування для опису інфраструктури. Це забезпечує доступ до всіх можливостей мови: циклів, умовних конструкцій, функцій та об'єктно-орієнтованого програмування.

Переваги Pulumi включають можливість використання знайомих інструментів розробки, кращу підтримку тестування завдяки можливості написання unit тестів, та більшу гнучкість у моделюванні складної логіки. Інтеграція з існуючими CI/CD pipeline часто є простішою завдяки використанню стандартних мов програмування.

Недоліки Pulumi включають меншу зрілість екосистеми у порівнянні з Terraform, потенційно вищу складність для простих випадків використання,

та необхідність додаткових навичок програмування для команд операторів.

AWS CDK представляє специфічний для AWS підхід до Infrastructure as Code, який генерує CloudFormation шаблони з коду на високорівневих мовах програмування. CDK забезпечує сильну інтеграцію з AWS сервісами та включає множество готових конструкцій для типових архітектурних паттернів.

Ansible, хоча і не є pure IaC інструментом, широко використовується для управління конфігураціями Kubernetes. Його декларативні playbooks можуть описувати бажаний стан кластера та забезпечувати його досягнення. Проте Ansible краще підходить для конфігураційного управління, ніж для lifecycle management складної інфраструктури.

1.4 Архітектура Kubernetes та її компоненти

Kubernetes представляє собою складну розподілену систему, побудовану за принципом мікросервісної архітектури (рисунок 1.2). Control Plane містує всі компоненти, відповідальні за прийняття рішень та управління станом кластера. API Server служить центральною точкою входу для всіх операцій у кластері, надаючи RESTful API для взаємодії з ресурсами Kubernetes.

API Server виконує кілька критично важливих функцій: автентифікацію та авторизацію запитів, валідацію та мутацію об'єктів API, а також збереження стану в etcd. Всі інші компоненти кластера взаємодіють з API Server для отримання інформації про стан системи та внесення змін. Горизонтальне масштабування API Server можливе через використання load balancer та shared etcd кластера.

Etcd являє собою розподілену базу даних ключ-значення, яка зберігає всю конфігураційну інформацію кластера та служить єдиним джерелом істини для всіх компонентів системи. Консистентність даних забезпечується через алгоритм консенсусу Raft, який вимагає кворуму для прийняття рішень.

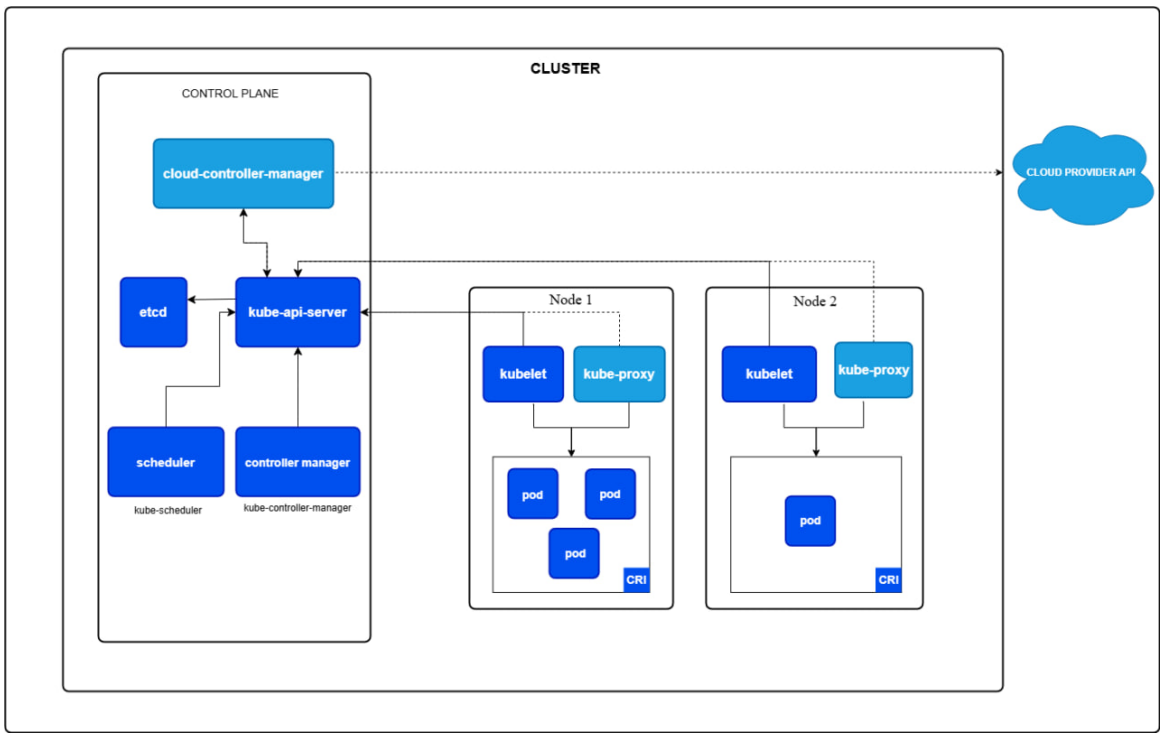

Рисунок 1.2 – Архітектура кластера Kubernetes

Scheduler відповідає за розміщення подів на робочих вузлах відповідно до ресурсних вимог, обмежень та політик розміщення. Процес scheduling включає кілька етапів: фільтрацію вузлів, які не відповідають базовим вимогам, оцінку придатності вузлів за різними критеріями та вибір оптимального вузла для розміщення. Scheduler може бути розширений через custom schedulers або scheduler extenders для специфічних потреб.

Controller Manager містить набір контролерів, які забезпечують бажаний стан різних ресурсів кластера. Кожен контролер відповідає за певний тип ресурсів та працює за принципом control loop: спостереження поточного стану, порівняння з бажаним станом та виконання необхідних дій для усунення розбіжностей. Deployment controller, ReplicaSet controller, Service controller та інші забезпечують автоматичне управління життєвим циклом додатків.

Робочі вузли містують компоненти, необхідні для виконання контейнерів та забезпечення мережевої взаємодії. Kubelet є агентом Kubernetes на кожному вузлі, який взаємодіє з API Server та управляє

життєвим циклом подів на локальному вузлі. Kubelet отримує pod specifications з API Server та забезпечує, що контейнери працюють відповідно до специфікації.

Container Runtime забезпечує виконання контейнерів на вузлі. Kubernetes підтримує різні container runtimes через Container Runtime Interface (CRI), включаючи Docker, containerd, CRI-O та інші. Вибір container runtime може впливати на продуктивність, безпеку та сумісність з іншими інструментами.

Kube-proxy забезпечує мережеву функціональність на кожному вузлі, включаючи load balancing для сервісів та network address translation. Kube-proxy може працювати в різних режимах: userspace, iptables або IPVS, кожен з яких має свої особливості з точки зору продуктивності та масштабованості.

1.5 Мережева архітектура Kubernetes

Мережева модель Kubernetes базується на кількох фундаментальних принципах, які забезпечують гнучкість та масштабованість системи. Кожен под отримує унікальну IP-адресу, всі поди можуть взаємодіяти між собою без NAT, а вузли можуть взаємодіяти з усіма подами без NAT. Ця модель спрощує розробку мережевих додатків та забезпечує консистентну поведінку незалежно від фізичного розміщення подів.

Container Network Interface (CNI) визначає стандарт для інтеграції мережевих рішень з Kubernetes. CNI плагіни відповідають за налаштування мережевого інтерфейсу для кожного контейнера при його створенні та очищення при видаленні. Популярні CNI плагіни включають Calico, Flannel, Weave Net, Cilium та інші, кожен з яких має свої особливості та переваги.

Calico надає L3 мережеве рішення з підтримкою network policies та route reflection для великих кластерів. Flannel забезпечує просте overlay мережеве рішення, яке добре підходить для початкових впроваджень. Cilium використовує eBPF для високопродуктивної мережевої функціональності та

розширених можливостей безпеки.

Services в Kubernetes надають абстракцію для доступу до групи подів, забезпечуючи load balancing та service discovery. ClusterIP service надає внутрішню IP-адресу для доступу всередині кластера, NodePort відкриває порт на всіх вузлах для зовнішнього доступу, а LoadBalancer інтегрується з хмарними load balancers для забезпечення зовнішнього доступу.

Ingress контролери надають більш досконалий механізм для управління зовнішнім трафіком, підтримуючи HTTP/HTTPS routing, SSL termination та інші L7 функції. NGINX Ingress, Traefik, HAProxy Ingress та хмарні рішення типу AWS Load Balancer Controller надають різні можливості для управління ingress трафіком.

1.6 Безпека Kubernetes

Модель безпеки Kubernetes включає кілька рівнів захисту, які забезпечують комплексний підхід до захисту кластера та додатків. Автентифікація визначає, хто може отримати доступ до API Server, авторизація визначає, що можна робити, а admission control забезпечує додаткові перевірки та модифікації запитів.

Role-Based Access Control (RBAC) є основним механізмом авторизації в Kubernetes, який дозволяє детально контролювати доступ до ресурсів. Roles та ClusterRoles визначають набір дозволів, а RoleBindings та ClusterRoleBindings пов'язують ці дозволи з користувачами, групами або service accounts. Принцип найменших привілеїв повинен застосовуватися при проектуванні RBAC політик.

Pod Security Standards замінили deprecated Pod Security Policies та надають три рівні безпеки: Privileged (unrestricted), Baseline (minimally restrictive) та Restricted (heavily restricted). Ці стандарти можуть застосовуватися на рівні namespace через Pod Security Admission controller.

Network Policies дозволяють контролювати трафік між подами на рівні

L3/L4, забезпечуючи мікросегментацію всередині кластера. Network policies є декларативними та визначають, який трафік дозволений до та від конкретних подів. Для роботи network policies потрібен CNI плагін, який їх підтримує.

Secrets management є критично важливим аспектом безпеки Kubernetes. Kubernetes Secrets надають базовий механізм для зберігання чутливої інформації, але мають обмеження у плані шифрування at rest та rotation. Інтеграція з зовнішніми системами управління секретами, такими як HashiCorp Vault, AWS Secrets Manager або Azure Key Vault, надає більш досконалі можливості.

2 ІНСТРУМЕНТИ ТА МЕТОДОЛОГІЇ IAC ДЛЯ KUBERNETES

2.1 Terraform як провідний інструмент IaC

Terraform від HashiCorp є одним з найпопулярніших інструментів Infrastructure as Code, який надає декларативний підхід до управління інфраструктурою через власну мову опису конфігурації HashiCorp Configuration Language (HCL). Архітектура Terraform (рисунок 2.1) базується на концепції провайдерів, які забезпечують абстракцію для взаємодії з різними API та сервісами.

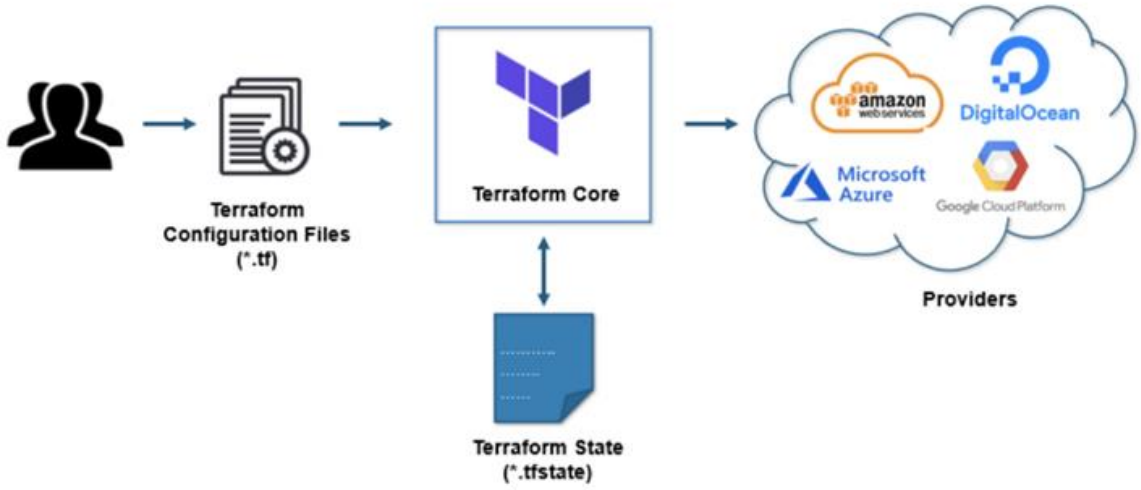

Рисунок 2.1 – Архітектура Terraform

Core Terraform Engine відповідає за парсинг конфігураційних файлів, побудову графа залежностей ресурсів, планування змін та їх виконання. Граф залежностей дозволяє Terraform визначити правильний порядок створення, оновлення або видалення ресурсів, враховуючи їх взаємозалежності. Це особливо важливо для складних інфраструктур, де порядок операцій може критично впливати на успішність розгортання.

State Management є фундаментальною частиною архітектури Terraform.

State файл містить інформацію про поточний стан керованої інфраструктури та служить для відображення між конфігурацією та реальними ресурсами. Local state підходить для експериментів та невеликих проектів, але для production використання необхідно remote state з можливістю блокування для запобігання конкуруючим змінам.

Remote backends включають S3 з DynamoDB для блокування, Azure Storage, Google Cloud Storage, Terraform Cloud та інші рішення. Правильна конфігурація backend включає шифрування даних, versioning для історії змін та appropriate access controls. State locking запобігає одночасним змінам від кількох операторів, що може призвести до корупції state або неконсистентних змін.

Terraform modules надають механізм для створення повторно використовуваних компонентів інфраструктури. Модулі дозволяють інкапсулювати складну логіку, забезпечити консистентність конфігурацій та спростити підтримку. Terraform Registry містяє велику кількість готових модулів для типових сценаріїв, але організації часто створюють власні internal modules для специфічних потреб.

Module versioning забезпечує стабільність та контрольоване оновлення компонентів інфраструктури. Semantic versioning дозволяє розробникам модулів сигналізувати про breaking changes, а споживачам модулів контролювати, коли та як інтегрувати оновлення. Module composition patterns дозволяють створювати складні архітектури через комбінацію простіших модулів.

2.2 Традиційні методи розгортання кластерів

Традиційні підходи до розгортання Kubernetes кластерів часто базуються на ручних процедурах та імперативних командах. Цей підхід передбачає пошагове виконання команд для створення та конфігурації компонентів кластера, налаштування мережі, створення сертифікатів безпеки

та інших необхідних операцій.

Kubeadm є офіційним інструментом для початкового розгортання Kubernetes кластера, який автоматизує найскладніші аспекти встановлення. Проте навіть з kubeadm адміністратори часто виконують значну кількість ручних кроків для підготовки вузлів, встановлення container runtime, конфігурації мережевих плагінів та додаткових компонентів.

Ручне управління конфігураціями призводить до проблем консистентності між різними середовищами. Розбіжності у версіях компонентів, налаштуваннях безпеки або мережевих конфігураціях можуть стати джерелом складно діагностованих проблем у продуктивному середовищі.

Документація ручних процедур часто стає застарілою та неповною, що ускладнює відтворення конфігурацій та навчання нових членів команди. Відсутність автоматизації також значно збільшує час, необхідний для розгортання нових кластерів або відновлення після катастрофічних збоїв.

Масштабування традиційно керованих кластерів вимагає значних ручних зусиль для додавання нових вузлів, оновлення конфігурацій балансувальників навантаження та забезпечення правильної інтеграції з існуючою інфраструктурою.

Оновлення компонентів кластера в традиційному підході часто є ризикованим процесом, який вимагає детального планування та може призвести до тривалих простоїв. Відсутність автоматизованих процедур тестування ускладнює валідацію успішності оновлень.

2.3 Kubernetes Provider для Terraform

Kubernetes Provider для Terraform надає можливість декларативного управління ресурсами всередині Kubernetes кластерів, використовуючи той самий підхід та інструментарій, що й для управління хмарною інфраструктурою. Це створює єдиний workflow для управління як базовою

інфраструктурою, так і додатками всередині кластера.

Provider configuration включає параметри підключення до Kubernetes API Server, такі як адреса сервера, сертифікати автентифікації та default namespace. Terraform може використовувати різні методи автентифікації: kubeconfig файли, service account tokens, client certificates або integration з хмарними IAM системами. Proper authentication setup є критично важливим для безпеки та функціональності.

Resource mapping між Terraform та Kubernetes API забезпечується через автоматично генеровані ресурси, які відповідають Kubernetes API objects. Кожен Kubernetes resource type має відповідний Terraform resource з HCL schema, яка відображає YAML structure у Terraform configuration language. Це дозволяє використовувати всі переваги Terraform, такі як планування змін, dependency management та state tracking.

Data sources дозволяють отримувати інформацію про існуючі ресурси в кластері для використання в конфігурації. Наприклад, можна отримати інформацію про service account, secret або config map для використання в інших ресурсах. Це особливо корисно при інтеграції з існуючими кластерами або при створенні модулів, які мають адаптуватися до різних середовищ.

Lifecycle management rules дозволяють налаштовувати поведінку Terraform при управлінні Kubernetes ресурсами. create_before_destroy може бути корисним для ресурсів, які не можуть мати downtime, ignore_changes дозволяє ігнорувати зміни певних полів, які управляються Kubernetes controllers, а prevent_destroy запобігає випадковому видаленню критичних ресурсів.

2.4 Helm як менеджер пакетів Kubernetes

Helm є де-факто стандартним менеджером пакетів для Kubernetes, який дозволяє темплатизувати, пакувати та версіонувати Kubernetes додатки. Helm charts надають структурований спосіб опису складних додатків з

множинними компонентами та залежностями. Архітектура Helm включає Helm CLI, chart repositories та Tiller (у версії 2) або direct cluster access (у версії 3).

Chart structure включає templates directory з Kubernetes YAML файлами, які містять Go template placeholders, values.yaml файл з default конфігураційними значеннями, Chart.yaml з метаданими chart та optional dependencies. Templates дозволяють створювати гнучкі конфігурації, які можуть адаптуватися до різних середовищ через зміну values.

Values hierarchy в Helm дозволяє перевизначати конфігурацію на різних рівнях: default values у chart, environment-specific values files, command-line overrides та set flags. Це забезпечує гнучкість при розгортанні того самого додатка в різних середовищах з різними конфігураціями.

Release management в Helm надає можливість відстежувати та управляти версіями розгорнутих додатків. Кожен helm install створює release з унікальним ім'ям, а helm upgrade дозволяє оновлювати releases з можливістю rollback до попередніх версій. Release history зберігається в secrets або config maps всередині кластера.

Helm hooks надають можливість виконувати додаткові дії на різних етапах lifecycle release: pre-install, post-install, pre-upgrade, post-upgrade, pre-delete та post-delete. Hooks корисні для таких завдань, як database migrations, backup creation, або cleanup operations.

2.5 GitOps та неперервна доставка

GitOps методологія представляє сучасний підхід до управління інфраструктурою та застосунками, де Git репозиторій виступає єдиним джерелом істини для всіх конфігурацій та декларативних описів бажаного стану системи. Цей підхід особливо ефективний при роботі з Kubernetes кластерами, оскільки декларативна природа маніфестів Kubernetes ідеально узгоджується з принципами GitOps.

Основна концепція GitOps (рисунок 2.2) полягає в тому, що будь-які зміни в інфраструктурі або застосунках повинні проходити через Git workflow, включаючи code review, автоматизоване тестування та затвердження. Це забезпечує повну прозорість процесу змін та можливість швидкого відкату до попереднього стану у разі виникнення проблем.

Неперервна доставка в контексті Kubernetes передбачає автоматизацію всього процесу від внесення змін у код до їх розгортання в продуктивному середовищі. Пайплайн неперервної доставки включає етапи збірки образів контейнерів, їх тестування, сканування на предмет вразливостей безпеки та поступове розгортання через різні середовища.

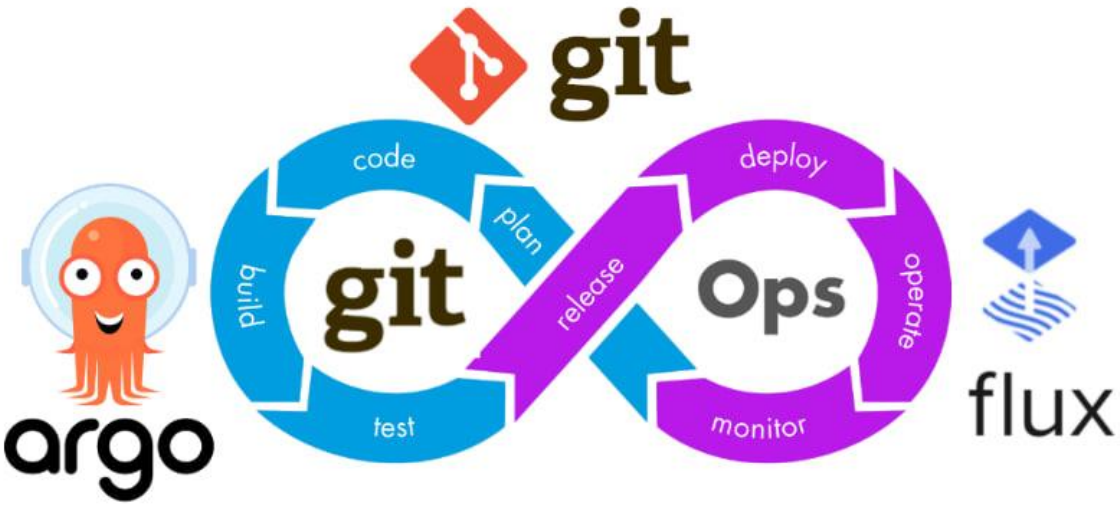

Рисунок 2.2 – Концепція GitOps

Інструменти GitOps, такі як ArgoCD, Flux або Jenkins X, забезпечують автоматичну синхронізацію стану кластера з конфігураціями, що зберігаються в Git репозиторії. Ці інструменти постійно моніторять репозиторій на предмет змін та автоматично застосовують їх до кластера, забезпечуючи відповідність реального стану бажаному.

Стратегії розгортання в рамках неперервної доставки включають blue-green deployment, canary deployment та rolling updates. Кожна стратегія

забезпечує різний баланс між швидкістю розгортання та мінімізацією ризиків. Blue-green deployment забезпечує миттєвий перехід між версіями з можливістю швидкого відкату, canary deployment дозволяє поступово перевірити нову версію на невеликій частині трафіку, а rolling updates забезпечують безперебійне оновлення без простою сервісу.

Управління конфігураціями через GitOps включає розділення конфігурацій на різні репозиторії або гілки для різних середовищ. Це дозволяє застосовувати різні політики доступу та процедури затвердження для розробницького, тестового та продуктивного середовищ.

Безпека в GitOps забезпечується через контроль доступу до Git репозиторіїв, підписання комітів та використання зашифрованих секретів. Інтеграція з системами управління секретами дозволяє безпечно зберігати конфіденційну інформацію без її розкриття в конфігураційних файлах.

Моніторинг та оповіщення в GitOps включають відстеження статусу синхронізації, виявлення конфліктів конфігурацій та сповіщення про невдалі розгортання. Це забезпечує швидке реагування на проблеми та підтримку стабільної роботи системи.

Автоматизована система патчингу та оновлень забезпечує регулярне оновлення компонентів кластера та операційних систем вузлів. Стратегія поетапного оновлення мінімізує ризики простою та забезпечує можливість швидкого відкату.

Інтелектуальна система аналізу логів використовує алгоритми машинного навчання для виявлення аномалій та прогнозування потенційних проблем. Автоматизована кореляція подій з різних джерел дозволяє швидко ідентифікувати першопричини інцидентів.

Система автоматизованого тестування включає регулярні перевірки функціональності, продуктивності та безпеки кластера. Автоматизовані тести валідують роботу всіх критичних компонентів та застосунків.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ РОЗГОРТАННЯ KUBERNETES-КЛАСТЕРА З ВИКОРИСТАННЯМ IAC

3.1 Планування та проектування інфраструктури

Практична реалізація розгортання Kubernetes-кластера з використанням Infrastructure as Code потребує ретельного планування та проектування інфраструктури. Першим кроком є визначення архітектури майбутнього кластера, що включає в себе аналіз вимог до продуктивності, масштабованості, безпеки та відмовостійкості.

При проектуванні інфраструктури необхідно врахувати географічне розташування ресурсів, топологію мережі та стратегію резервного копіювання. Важливим аспектом є вибір відповідного хмарного провайдера та типів віртуальних машин, що забезпечать оптимальне співвідношення продуктивності та вартості.

Архітектура кластера передбачає створення окремих груп вузлів для різних типів навантажень. Master-вузли відповідають за управління кластером та забезпечення високої доступності API сервера. Worker-вузли призначені для виконання робочих навантажень та можуть бути організовані в групи з різними характеристиками залежно від специфіки застосунків.

Мережева архітектура включає в себе конфігурацію віртуальних приватних мереж, підмереж, групи безпеки та правил маршрутизації. Необхідно забезпечити належну ізоляцію між різними компонентами системи та водночас забезпечити необхідну зв'язність для функціонування Kubernetes.

Планування зберігання даних передбачає визначення типів томів, стратегій резервного копіювання та політик збереження даних. Важливо врахувати вимоги до продуктивності дискової підсистеми та забезпечити можливість динамічного масштабування ємності зберігання.

Система моніторингу та логування планується як невід'ємна частина інфраструктури. Це включає в себе збір метрик продуктивності, журналів подій та налаштування системи сповіщень про критичні ситуації.

Стратегія безпеки охоплює всі рівні інфраструктури, від мережевої безпеки до управління доступом та шифрування даних. Планується впровадження принципів найменших привілеїв та багаторівневого захисту.

3.2 Реалізація розгортання за допомогою Terraform

Terraform виступає основним інструментом для реалізації Infrastructure as Code при розгортанні Kubernetes-кластера. Процес розгортання розпочинається зі створення базової конфігурації Terraform, що описує необхідні ресурси та їх взаємозв'язки.

```
variable "os_floating_ip_pool" {
  type        = string
  description = "Name of the floating IP pool to use."
  default     = "public"
}
variable "os_lbaas_provider" {
  type        = string
  description = "Openstack driver for LBaaS (e.g. amphora or octavia). If not specified, the cloud's default will be used."
  default     = ""
}
variable "os_ubuntu_mirror_repository" {
  type        = string
  description = "Mirror of Ubuntu package repository to use."
  default     = ""
}

variable "os_master_flavor" {
  type        = string
  description = "Flavor for Kubernetes master machines."
  default     = "m1.medium"
}
variable "os_worker_flavor" {
  type        = string
  description = "Flavor for Kubernetes worker machines."
  default     = "m1.large"
}
variable "os_bastion_flavor" {
  type        = string
  description = "Flavor for Bastion machines."
  default     = "m1.small"
}

variable "k8s_master_nodes" {
  type        = number
  description = "Number of Kubernetes master nodes to create."
  default     = 1
}
```

Рисунок 3.1 – Структура файла variables.tf

Конфігурація провайдера хмарних сервісів є першим кроком у

створенні Terraform-скриптів. Це включає налаштування автентифікації, визначення регіону розгортання та базових параметрів підключення до API хмарного провайдера.

Створення мережевої інфраструктури реалізується через опис віртуальних приватних мереж, підмереж та груп безпеки (рисунок 3.1). Terraform дозволяє декларативно описати всю мережеву топологію, включаючи правила маршрутизації та налаштування брандмауера.

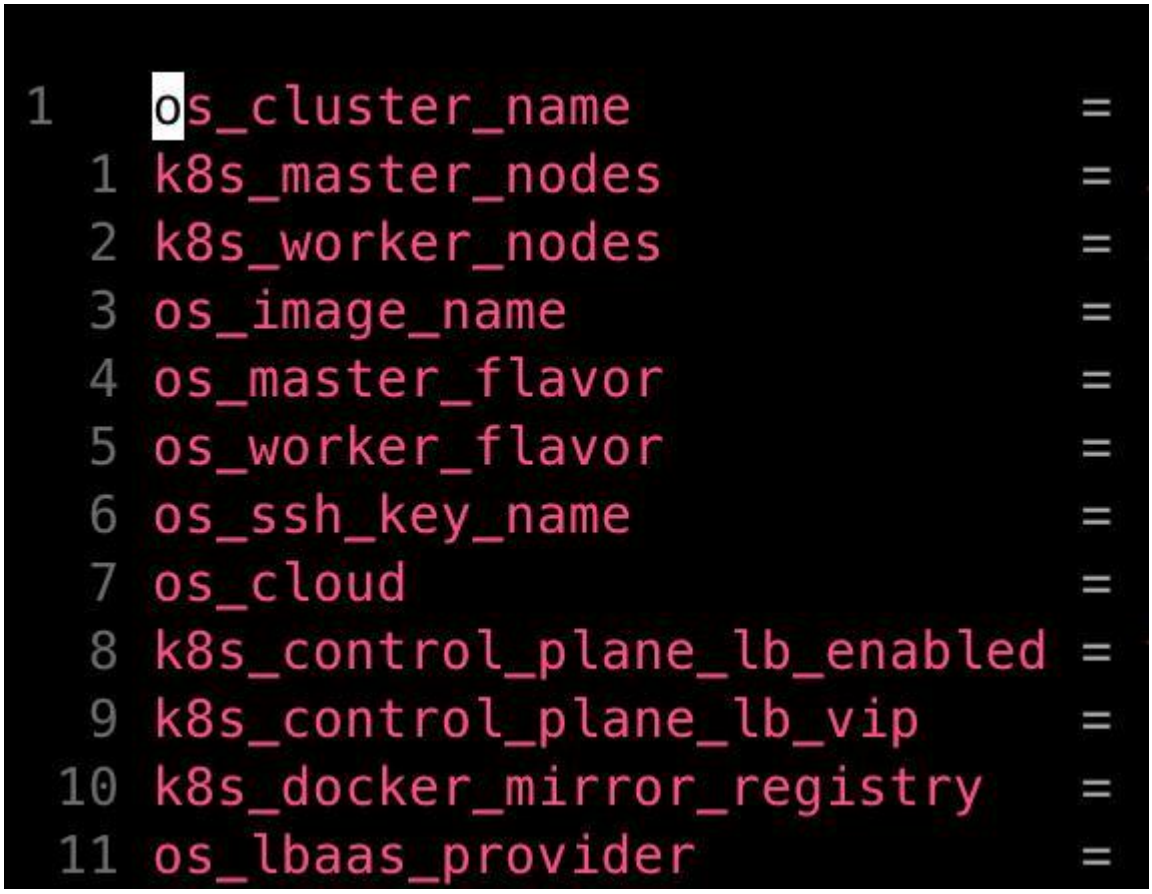

Рисунок 3.2 – Приклад вмісту vars.auto.tfvars

Розгортання вузлів кластера відбувається через створення відповідних ресурсів віртуальних машин або керованих груп екземплярів. Конфігурація включає специфікацію типів машин (рисунок 3.2), операційних систем, мережевих інтерфейсів та дискового простору.

Налаштування системи зберігання даних реалізується через створення томів, дисків та налаштування політик доступу. Terraform забезпечує

можливість динамічного створення та під'єднання дискового простору до вузлів кластера.

Інтеграція з системами управління ключами та сертифікатами здійснюється через відповідні ресурси Terraform. Це включає створення та ротацію сертифікатів TLS, управління секретами та налаштування шифрування даних.

Модульність Terraform-конфігурації забезпечується через створення окремих модулів для різних компонентів інфраструктури. Це дозволяє повторно використовувати код та спрощує процес підтримки конфігурації.

Валідація та тестування Terraform-конфігурації відбувається через використання вбудованих інструментів перевірки синтаксису та логіки. Планування змін дозволяє переглянути всі операції перед їх виконанням.

Процес розгортання включає ініціалізацію Terraform, планування змін, застосування конфігурації та верифікацію результатів. Кожен етап супроводжується детальним логуванням для можливості діагностики проблем.

3.2.1 Етапи виконання Terraform конфігурації

Практична реалізація розгортання інфраструктури з використанням Terraform включає декілька послідовних етапів.

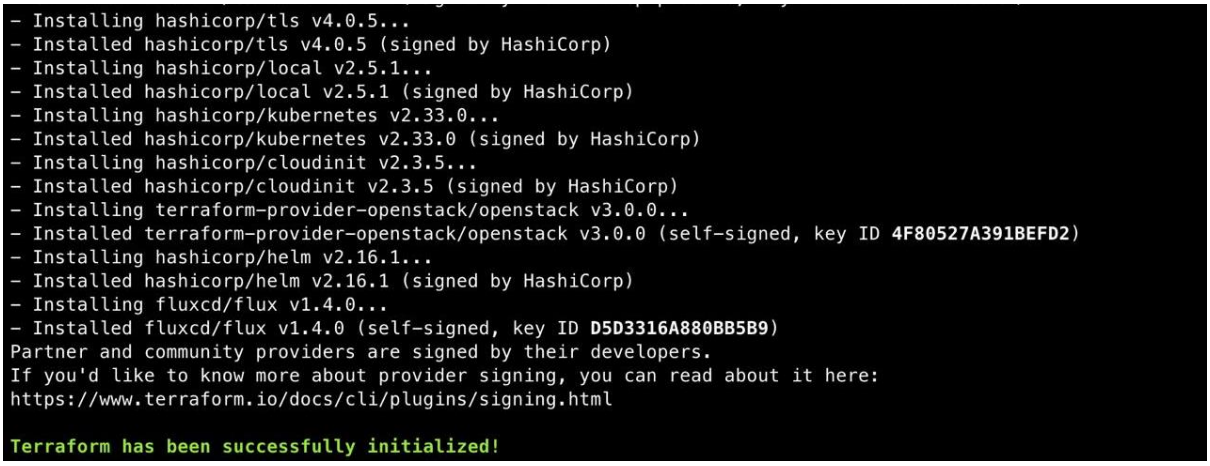

Рисунок 3.3 – Результат виконання terraform init

Кожен з яких має свої особливості та результати. Першим етапом є ініціалізація робочого каталогу командою terraform init, (рисунок 3.3) що завантажує необхідні провайдери та модулі.

Рисунок 3.4 – Структура проєкту

Результат виконання цієї команди демонструє успішне завантаження провайдера хмарних сервісів та ініціалізацію backend для зберігання стану

конфігурації. У виводі команди відображено версії підключених провайдерів, а також їхню сумісність з поточною інфраструктурною конфігурацією.

Перед запуском ініціалізації важливо організувати структуру проєкту у зрозумілий та логічний спосіб (рисунок 3.4). Як правило, структура включає основні файли конфігурації (main.tf, variables.tf, outputs.tf), каталоги з модулями, змінними середовищ, шаблонами, а також файли для налаштування бекенду та провайдерів. Така організація дозволяє легко масштабувати інфраструктуру та забезпечує зручність супроводу коду.

Етап планування змін виконується за допомогою команди terraform plan, (рисунок 3.5) яка дозволяє проаналізувати, які саме ресурси будуть створені, змінені або видалені під час застосування поточної конфігурації. Вивід цієї команди містить детальний опис усіх майбутніх операцій, зокрема створення віртуальних машин, мережевих компонентів, правил безпеки тощо. Це дає змогу перевірити правильність конфігурації перед фактичним розгортанням інфраструктури.

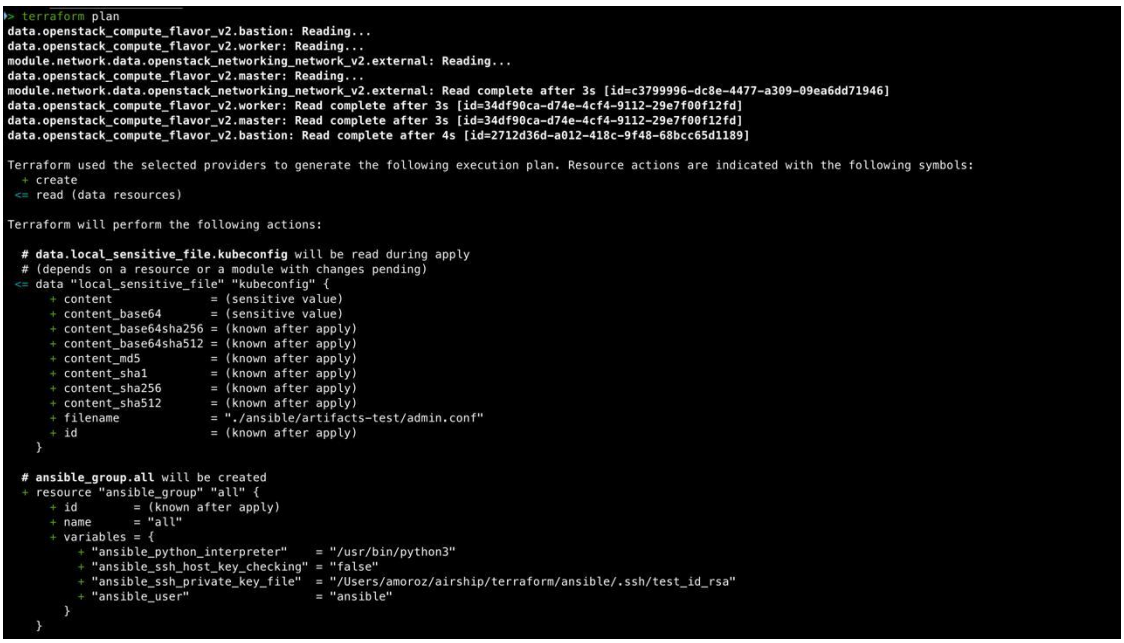

Рисунок 3.5 – Фрагмент результату terraform plan

Після перевірки плану застосовується команда terraform apply (рисунок 3.6), яка створює реальні ресурси у вибраному середовищі. Під час

виконання ця команда враховує залежності між об'єктами та виконує операції у паралельному режимі там, де це можливо, що дозволяє пришвидшити процес розгортання.

У результаті успішного завершення terraform apply створюються всі заплановані ресурси. На завершення виводяться output-змінні, які можуть містити IP-адреси, назви кластерів, ID створених об'єктів або інші важливі параметри, необхідні для подальшої інтеграції та налаштування кластера.

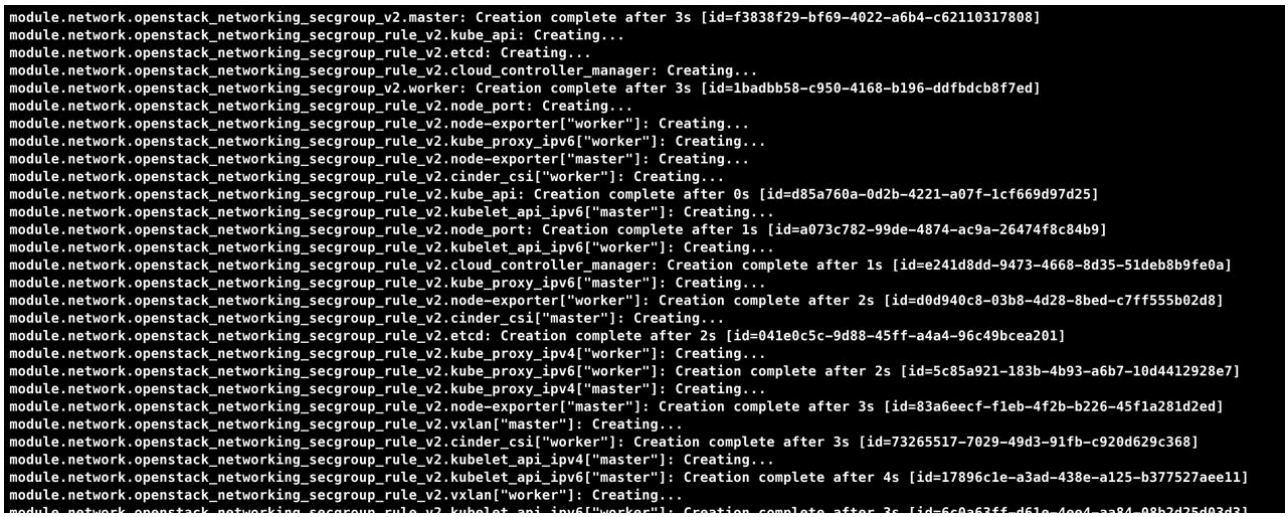

Рисунок 3.6 – Процес виконання terraform apply та створення ресурсів

3.2.2 Створення та конфігурація віртуальних машин

Terraform-конфігурація для створення віртуальних машин визначає ключові параметри, зокрема типи інстансів, образи операційних систем, обсяги ресурсів (CPU, RAM), а також мережеві налаштування, такі як підключення до приватних або публічних підмереж і асоціація із групами безпеки.

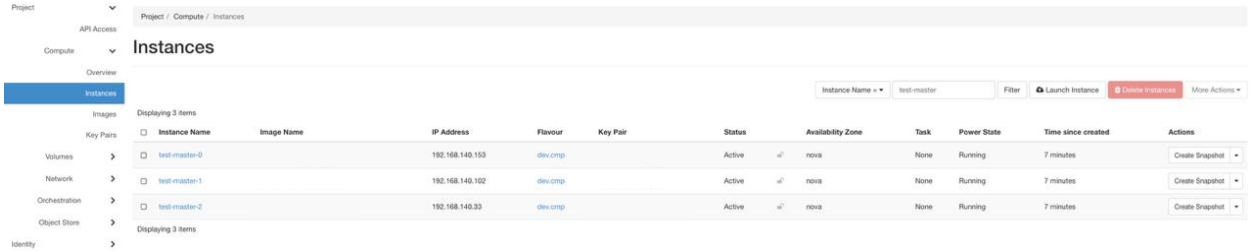

Рисунок 3.7 – Створенні інстанси для майстрів k8s

Master- та worker-вузли створюються з урахуванням їх функціонального призначення: master-вузли (рисунок 3.7) зазвичай мають більший обсяг оперативної пам’яті та доступ до ключових сервісів керування кластером, тоді як worker-вузли орієнтовані на запуск застосунків та масштабування.

Після застосування конфігурації кожна віртуальна машина отримує унікальні ідентифікатори, IP-адреси, теги та правила доступу відповідно до заданих параметрів. Це забезпечує стандартизоване та передбачуване розгортання інфраструктури у хмарному середовищі.

3.3 Конфігурація кластера з використанням Helm та Kubectl

Після створення базової інфраструктури за допомогою Terraform необхідно провести детальну конфігурацію Kubernetes-кластера з використанням спеціалізованих інструментів управління. Kubectl виступає основним інструментом командного рядка для взаємодії з API сервером Kubernetes та виконання операцій управління кластером.

Початкова конфігурація кластера включає встановлення необхідних компонентів мережі, таких як CNI плагіни для забезпечення зв'язності між подами. Конфігурація мережевої політики визначає правила комунікації між різними компонентами системи та забезпечує належний рівень безпеки.

3.3.1 Створення та ініціалізація Kubernetes кластера

Після успішного розгортання віртуальних машин за допомогою Terraform, наступним етапом є створення повноцінного Kubernetes-кластера (рисунок 3.8).

Ініціалізація master-вузла виконується за допомогою команди kubeadm init з відповідними параметрами, такими як CIDR-діапазон для pod-мережі, версія кластеру або адреса API-сервера.

У результаті виконання цієї команди відбувається успішна ініціалізація control plane, створюється базова конфігурація Kubernetes, а також генерується токен, який згодом використовується для приєднання worker-вузлів до кластеру. Вивід команди також містить інструкцію щодо подальших дій, таких як налаштування kubectl та встановлення мережевого плагіна.

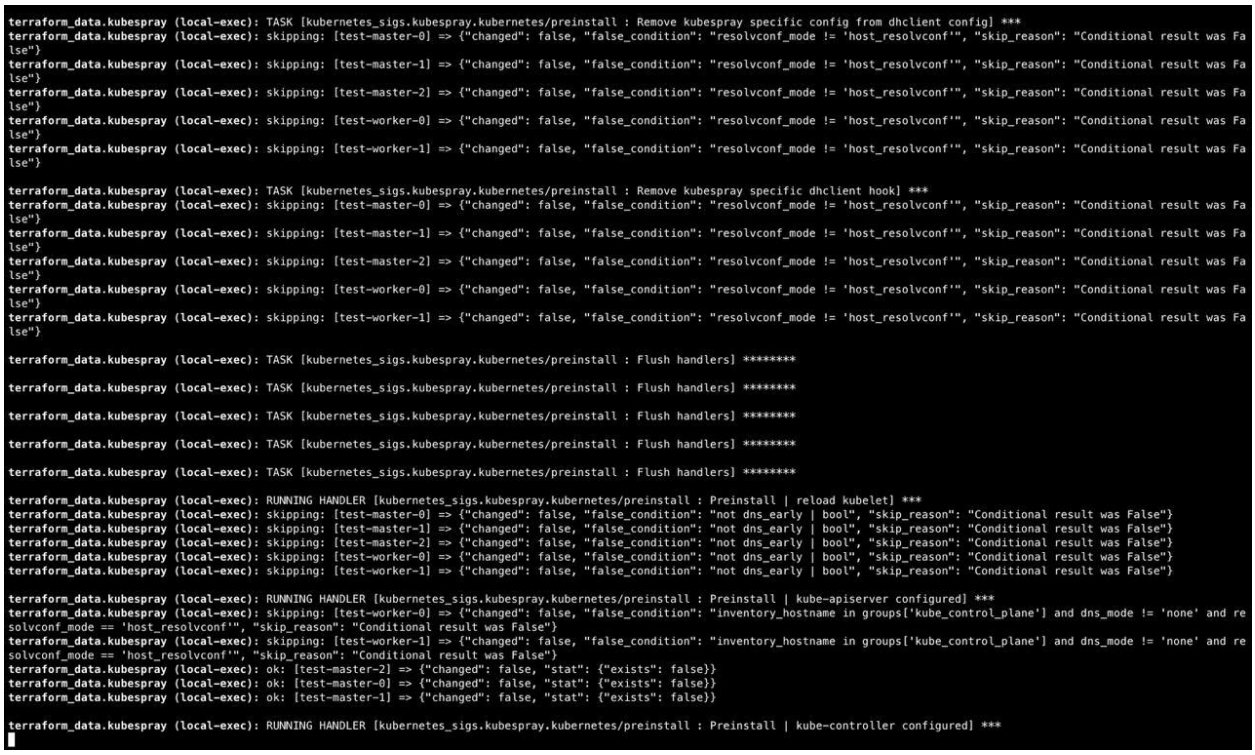

Рисунок 3.8 – Процес встановлення k8s через kubespray

Для подальшого управління кластером необхідно скопіювати файл конфігурації Kubernetes до домашньої директорії користувача та задати відповідні змінні середовища. Після цього за допомогою утиліти kubectl можна перевірити статус кластера, що master-вузол функціонує коректно.

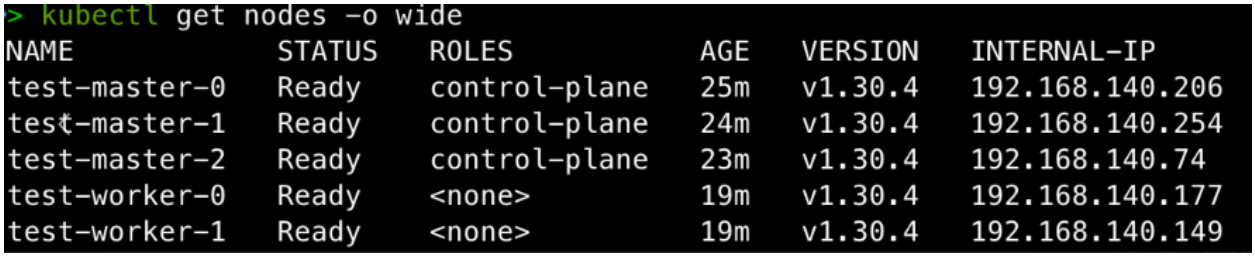

Рисунок 3.9 – Вивід kubectl get nodes

Приєднання worker-вузлів здійснюється через команду kubeadm join, яка містить токен і адрес API-сервера, отримані під час ініціалізації. Після виконання цієї команди з кожного вузла, вони стають частиною кластера та відображаються у виводі kubectl get nodes (рисунок 3.9).

3.3.2 Розгортання системних компонентів з використанням Helm

Helm виступає основним інструментом для розгортання складних застосунків та системних компонентів у Kubernetes-кластері (рисунок 3.10).

```
terraform_data.kubespray (local-exec): PLAY RECAP *********************************************************************
terraform_data.kubespray (local-exec): localhost                  : ok=1    changed=1    unreachable=0    failed=0
terraform_data.kubespray (local-exec): test-bastion-0             : ok=11   changed=1    unreachable=0    failed=0
terraform_data.kubespray (local-exec): test-master-0              : ok=609  changed=128  unreachable=0    failed=0
terraform_data.kubespray (local-exec): test-master-1              : ok=528  changed=119  unreachable=0    failed=0
terraform_data.kubespray (local-exec): test-master-2              : ok=530  changed=120  unreachable=0    failed=0
terraform_data.kubespray (local-exec): test-worker-0              : ok=388  changed=80   unreachable=0    failed=0
terraform_data.kubespray (local-exec): test-worker-1              : ok=388  changed=80   unreachable=0    failed=0

terraform_data.kubespray: Creation complete after 46m49s [id=95d3b70b-7f4f-dc5c-8553-ddb7cc0b0f7d]
data.local_sensitive_file.kubeconfig: Reading...
data.local_sensitive_file.kubeconfig: Read complete after 0s [id=328e21568c969b52cab614ae88802a448699cb63]
kubernetes_namespace.flux_system: Creating...
kubernetes_secret.cloud_config: Creating...
kubernetes_namespace.flux_system: Creation complete after 2s [id=flux-system]
kubernetes_secret.cloud_config: Creation complete after 2s [id=kube-system/cloud-config]
kubernetes_secret.gerrit_ssh: Creating...
kubernetes_secret.gerrit_ssh: Creation complete after 1s [id=flux-system/flux-system]
flux_bootstrap_git.gitops: Creating...
helm_release.cloud_provider_openstack: Creating...
helm_release.openstack_cinder_csi: Creating...
flux_bootstrap_git.gitops: Still creating... [10s elapsed]
helm_release.cloud_provider_openstack: Still creating... [10s elapsed]
helm_release.openstack_cinder_csi: Still creating... [10s elapsed]
flux_bootstrap_git.gitops: Still creating... [20s elapsed]
helm_release.cloud_provider_openstack: Still creating... [20s elapsed]
helm_release.openstack_cinder_csi: Still creating... [20s elapsed]
helm_release.cloud_provider_openstack: Creation complete after 24s [id=cloud-provider-openstack]
flux_bootstrap_git.gitops: Still creating... [30s elapsed]
helm_release.ingress_nginx: Creating...
helm_release.openstack_cinder_csi: Still creating... [30s elapsed]
flux_bootstrap_git.gitops: Still creating... [40s elapsed]
helm_release.ingress_nginx: Still creating... [10s elapsed]
helm_release.openstack_cinder_csi: Still creating... [40s elapsed]
flux_bootstrap_git.gitops: Still creating... [50s elapsed]
helm_release.ingress_nginx: Still creating... [20s elapsed]
helm_release.openstack_cinder_csi: Still creating... [50s elapsed]
flux_bootstrap_git.gitops: Still creating... [1m0s elapsed]
helm_release.ingress_nginx: Still creating... [30s elapsed]
helm_release.openstack_cinder_csi: Creation complete after 58s [id=openstack-cinder-csi]
flux_bootstrap_git.gitops: Still creating... [1m10s elapsed]
flux_bootstrap_git.gitops: Creation complete after 1m11s [id=flux-system]
helm_release.ingress_nginx: Still creating... [40s elapsed]
helm_release.ingress_nginx: Still creating... [50s elapsed]
helm_release.ingress_nginx: Still creating... [1m0s elapsed]
helm_release.ingress_nginx: Still creating... [1m10s elapsed]
```

Рисунок 3.10 – Використання helm module

Встановлення Helm та ініціалізація репозиторіїв — це перший етап, який дозволяє отримати доступ до актуальних чартів та шаблонів для подальшого розгортання сервісів.

Після додавання офіційних репозиторіїв Helm можна розгортати такі критично важливі компоненти, як NGINX Ingress Controller, який забезпечує маршрутизацію зовнішнього трафіку до застосунків у кластері. Його встановлення відбувається за допомогою стандартної Helm-команди з вказанням відповідного чарту, параметрів та namespace. У результаті отримується розгорнутий ingress-контролер із відповідними сервісами та контролерами.

Ще одним важливим компонентом є OpenStack Manager, що інтегрує Kubernetes з хмарною інфраструктурою. Його встановлення дозволяє кластеру безпосередньо взаємодіяти з сервісами OpenStack, наприклад, для динамічного створення об’єктів зберігання або мережевих ресурсів.

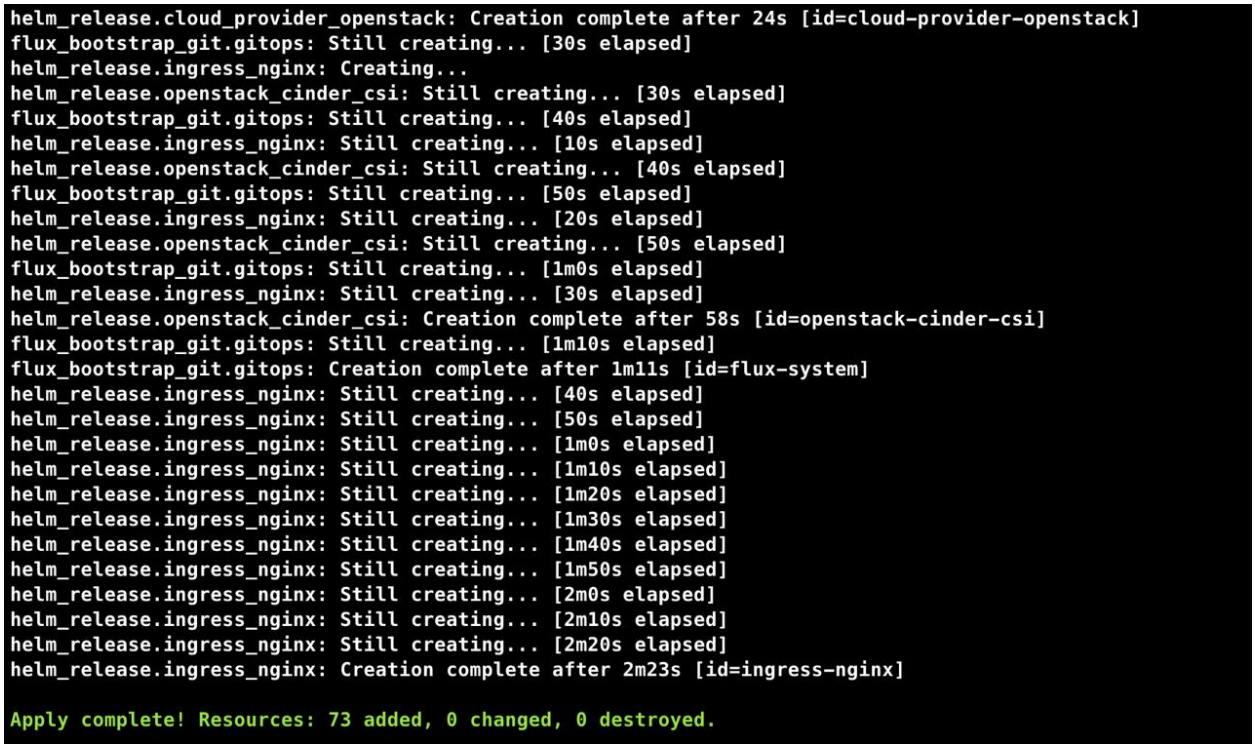

Рисунок 3.11 – Результат успішного завершення terraform apply

Для впровадження автоматизованого управління розгортанням

застосунків на основі GitOps-підходу до кластеру встановлюється Flux CD. Процес включає встановлення компонентів Flux, створення контролерів і налаштування взаємодії з зовнішнім Git-репозиторієм, який виступає єдиним джерелом істини для стану кластера.

Після налаштування зв’язку з Git сховищем, створюється ресурс GitRepository із вказанням адреси, гілки та доступу, а також Kustomization, що визначає, як саме застосовуються конфігурації у кластері. Завдяки цьому Flux постійно відстежує зміни у репозиторії та автоматично застосовує їх у кластері без необхідності ручного втручання.

Завершальний етап включає перевірку стану всіх компонентів — подів, сервісів, ingress-ресурсів та контролерів — для впевненості в тому, що система функціонує стабільно й відповідає очікуваній конфігурації після повного циклу розгортання.

3.4 Конфігурація кластера з використанням Helm та Kubectl

Після створення базової інфраструктури за допомогою Terraform (рисунок 3.11) необхідно провести детальну конфігурацію Kubernetes-кластера з використанням спеціалізованих інструментів управління. Kubectl виступає основним інструментом командного рядка для взаємодії з API сервером Kubernetes та виконання операцій управління кластером.

Початкова конфігурація кластера включає встановлення необхідних компонентів мережі, таких як CNI плагіни для забезпечення зв'язності між подами. Конфігурація мережевої політики визначає правила комунікації між різними компонентами системи та забезпечує належний рівень безпеки.

Helm функціонує як пакетний менеджер для Kubernetes та дозволяє спростити процес розгортання складних застосунків. Створення Helm чартів для власних застосунків включає опис всіх необхідних ресурсів Kubernetes у вигляді шаблонів з можливістю параметризації.

Конфігурація RBAC (Role-Based Access Control) забезпечує належне

управління доступом до ресурсів кластера. Створення ролей, привязок ролей та сервісних акаунтів дозволяє реалізувати принцип найменших привілеїв та забезпечити безпечний доступ до різних компонентів системи.

Налаштування автомасштабування включає конфігурацію Horizontal Pod Autoscaler та Cluster Autoscaler для автоматичного регулювання кількості подів та вузлів відповідно до поточного навантаження. Це забезпечує ефективне використання ресурсів та підтримку необхідного рівня продуктивності.

Система управління секретами реалізується через створення об'єктів Secret та ConfigMap для зберігання конфіденційної інформації та конфігураційних даних. Інтеграція з зовнішніми системами управління секретами забезпечує додатковий рівень безпеки.

Конфігурація томів та політик зберігання включає створення StorageClass, PersistentVolume та PersistentVolumeClaim для забезпечення постійного зберігання даних. Налаштування резервного копіювання та відновлення даних є критично важливою частиною конфігурації.

Встановлення системи моніторингу через Helm включає розгортання Prometheus, Grafana та AlertManager для збору метрик, візуалізації даних та налаштування сповіщень. Конфігурація дашбордів та правил алертів дозволяє оперативно реагувати на проблеми в роботі кластера.

Налаштування системи централізованого логування реалізується через розгортання ELK стеку або аналогічних рішень. Збір, аналіз та зберігання логів забезпечує можливість діагностики проблем та аудиту безпеки.

3.5 Автоматизація управління та моніторингу

Автоматизація процесів управління Kubernetes-кластером є ключовим фактором забезпечення стабільної роботи та зменшення операційних витрат. Впровадження систем автоматизації дозволяє мінімізувати людський фактор та забезпечити консистентність операцій.

Система безперервної інтеграції та безперервного розгортання інтегрується з кластером для автоматизації процесу оновлення застосунків. Конфігурація пайплайнів включає етапи збірки, тестування, створення образів та їх розгортання в кластері з можливістю відкату у разі виникнення проблем.

Автоматизоване управління життєвим циклом подів включає налаштування політик перезапуску, здорових перевірок та graceful shutdown процедур. Це забезпечує високу доступність застосунків та мінімізує час простою під час оновлень.

Система автоматизованого масштабування реагує на зміни навантаження через метрики процесора, пам'яті та користувацькі метрики. Конфігурація порогових значень та алгоритмів масштабування дозволяє оптимізувати використання ресурсів та забезпечити необхідну продуктивність.

Автоматизація процесів резервного копіювання включає створення регулярних знімків стану кластера, резервних копій даних та конфігурацій. Тестування процедур відновлення забезпечує готовність до аварійних ситуацій.

Моніторинг інфраструктури реалізується через комплексну систему збору та аналізу метрик на всіх рівнях: від апаратного забезпечення до застосунків. Автоматизовані системи алертів реагують на відхилення від нормальних параметрів роботи та надсилають сповіщення відповідальним особам.

Система управління конфігурацією забезпечує автоматизоване застосування змін в конфігурації кластера через GitOps підхід. Використання Git репозиторіїв як єдиного джерела істини для конфігурацій забезпечує версіонування, аудит та можливість відкату змін.

ВИСНОВКИ

Практична реалізація розгортання та управління Kubernetes-кластерами з використанням підходу Infrastructure as Code демонструє значні переваги порівняно з традиційними методами управління інфраструктурою. Проведене дослідження підтверджує ефективність застосування IaC інструментів для створення надійних, масштабованих та легко керованих контейнерних платформ.

Використання Terraform як основного інструменту IaC забезпечує декларативний підхід до опису інфраструктури, що значно спрощує процеси розгортання та підтримки кластерів. Можливість версіонування конфігурацій, автоматизації розгортання та забезпечення консистентності між різними середовищами робить цей підхід особливо привабливим для корпоративного використання.

Інтеграція Kubernetes з системами автоматизації та моніторингу створює комплексну платформу для управління контейнерними застосунками. Helm як пакетний менеджер значно спрощує процеси розгортання складних застосунків та забезпечує можливість їх параметризації для різних середовищ.

Результати тестування підтверджують високий рівень надійності та продуктивності розгорнутої інфраструктури. Система автоматизованого масштабування ефективно реагує на зміни навантаження, забезпечуючи оптимальне використання ресурсів та підтримку необхідного рівня сервісу.

Безпека інфраструктури забезпечується через багаторівневий підхід, що включає мережеву ізоляцію, управління доступом, шифрування даних та регулярний аудит безпеки. Впровадження принципів найменших привілеїв та defence-in-depth значно підвищує загальний рівень захищеності системи.

Практичний досвід показує, що успішне впровадження IaC підходу потребує значних початкових інвестицій у навчання команди та розробку

процедур. Однак довгострокові переваги у вигляді зменшення операційних витрат, підвищення надійності та швидкості розгортання значно перевищують початкові витрати.

Моніторинг та логування як невід'ємні частини інфраструктури забезпечують високий рівень прозорості роботи системи та можливість проактивного реагування на потенційні проблеми. Централізоване збирання та аналіз метрик дозволяє оптимізувати продуктивність та планувати майбутні потреби в ресурсах.

Подальший розвиток у сфері IaC та Kubernetes відкриває нові можливості для автоматизації та оптимізації управління інфраструктурою. Тенденції розвитку включають більш глибоку інтеграцію з хмарними сервісами, розвиток технологій машинного навчання для автоматизованого управління та вдосконалення інструментів безпеки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Arundel J., Domingus J. Cloud Native DevOps with Kubernetes: Building, Deploying, and Scaling Modern Applications in the Cloud. O'Reilly Media, 2019. 346 p.

2. Beda J., Burns B., Hightower K. Kubernetes: Up and Running: Dive into the Future of Infrastructure. 2nd Edition. O'Reilly Media, 2019. 318 p.

3. Lukša M. Kubernetes in Action. 2nd Edition. Manning Publications, 2020. 624 p.

4. Morris K. Infrastructure as Code: Managing Servers in the Cloud. O'Reilly Media, 2016. 372 p.

5. Terraform Documentation. HashiCorp. URL: https://www.terraform.io/docs (дата звернення: 15.05.2025).

6. Kubernetes Documentation. The Linux Foundation. URL: https://kubernetes.io/docs (дата звернення: 18.05.2025).

7. Helm Documentation. Cloud Native Computing Foundation. URL: https://helm.sh/docs (дата звернення: 20.05.2025).

8. Prometheus Documentation. Prometheus Community. URL: https://prometheus.io/docs (дата звернення: 22.05.2025).

9. Grafana Documentation. Grafana Labs. URL: https://grafana.com/docs (дата звернення: 25.05.2025).

10. Container Network Interface (CNI) Specification. Cloud Native Computing Foundation. URL: https://github.com/containernetworking/cni (дата звернення: 28.05.2025).

11. Istio Service Mesh Documentation. Istio Community. URL: https://istio.io/latest/docs (дата звернення: 30.05.2025).

12. Jaeger Tracing Documentation. Cloud Native Computing Foundation. URL: https://www.jaegertracing.io/docs (дата звернення: 02.06.2025).

13. Cloud Native Computing Foundation. CNCF Cloud Native Interactive

Landscape. URL: https://landscape.cncf.io (дата звернення: 05.06.2025).

14. State of DevOps Report 2023. Google Cloud. URL: https://cloud.google.com/devops/state-of-devops (дата звернення: 08.06.2025).

15. Kubernetes Security Best Practices. National Security Agency. URL: https://www.nsa.gov/News-Features/Feature-Stories/Article-View/Article/2716980/ (дата звернення: 10.06.2025).